

A Roadmap for Automating Lineage Tracing to Aid Automatically Explaining Machine Learning Predictions for Clinical Decision Support

Gang Luo, PhD

Department of Biomedical Informatics and Medical Education, University of Washington, UW Medicine South Lake Union, 850 Republican Street, Building C, Box 358047, Seattle, WA 98195, USA
luogang@uw.edu

Corresponding author:

Gang Luo, PhD

Department of Biomedical Informatics and Medical Education, University of Washington, UW Medicine South Lake Union, 850 Republican Street, Building C, Box 358047, Seattle, WA 98195, USA

Phone: 1-206-221-4596

Fax: 1-206-221-2671

Email: luogang@uw.edu

Abstract

Using machine learning predictive models for clinical decision support has great potential to improve patient outcomes and reduce healthcare costs. However, most machine learning models are black boxes that do not explain their predictions, forming a barrier to clinical adoption. To overcome this barrier, we recently developed an automated method to provide rule-style explanations of any machine learning model’s predictions on tabular data and to suggest customized interventions. Each explanation delineates the association between a feature value pattern and an outcome value. Although the association and intervention information is useful, the user of the automated explaining function often requires more detailed information to better understand the patient’s situation and to aid decision making. More specifically, consider a feature value in the explanation that is computed by an aggregation function on the raw data, such as the number of emergency department visits related to asthma that the patient had in the prior 12 months. The user often wants to rapidly drill through to see certain parts of the related raw data that produce the feature value. This task is frequently difficult and time-consuming because the few pieces of related raw data are submerged by many pieces of raw data of the patient unrelated to the feature value. To address this issue, this paper outlines an automated lineage tracing approach that adds automated drill-through capability to the automated explaining function, providing a roadmap for future research.

Keywords: clinical decision support; database management systems; forecasting; machine learning

Introduction

Machine learning has won almost all data science competitions [1] and is a hot topic these days. It is about computer algorithms that automatically learn from data, such as extreme gradient boosting, support vector machine, and random forest [2]. Using machine learning predictive models for clinical decision support has great potential to improve patient outcomes and reduce healthcare costs [3-10]. However, most machine learning models are black boxes that do not explain their predictions. This creates a barrier to clinical adoption. To overcome this barrier, we recently developed an automated method to offer rule-style explanations of any machine learning model’s predictions on tabular data and to suggest customized interventions without reducing the model’s performance measures [11-14]. Each rule-style explanation delineates the association between a feature value pattern and an outcome value. A feature is also called an independent variable. For the prediction of future emergency department (ED) visit or inpatient stay for asthma on an asthma patient, 1 example explanation is:

- The patient had 2 ED visits related to asthma in the prior 12 months

AND the patient’s average respiratory rate recorded in the prior 12 months is >25 and ≤ 28 breaths per minute

→ the patient will likely have at least 1 ED visit or inpatient stay for asthma in the next 12 months [13,14].

An ED visit is related to asthma if the ED visit has an asthma diagnosis code. For the item in the explanation showing that the patient had 2 ED visits related to asthma in the prior 12 months, 1 intervention suggested by our method is to apply control procedures that decrease the likelihood the patient will need emergency care.

The association and intervention information provided by our automatic explanation method for machine learning predictions is useful. However, the user of the automated explaining function often requires more detailed information to better understand the patient’s situation and to aid decision making. More specifically, consider a feature value on the left hand side of a rule-style explanation that is computed by an aggregation function on the raw data. The user often wants to rapidly drill through to see certain parts of the related raw data producing the feature value. In the context of a relational database, these parts refer to the most relevant attributes of the most essential source tuples producing the feature value. Which attributes are most relevant and which source tuples are most essential depend on both the concrete feature type and the clinical decision support application’s need, and are illustrated by several examples throughout the paper. The patterns embedded in these parts could provide additional information on the patient that was lost during the aggregation process to compute the feature value. This drill-through task is frequently difficult and time-consuming because the few pieces of related raw data are submerged by many pieces of raw data of the patient unrelated to the feature value. For example, as Table 1 shows, the list of encounters of an asthma patient displayed on the standard interface of an electronic medical record system includes much information that is irrelevant to the feature value “2 of the number of ED visits related to asthma that the patient had in the prior 12 months.”

Table 1. An example list of encounters of an asthma patient displayed on the standard interface of an electronic medical record system. The grayed “Primary diagnosis” column does not show up on the standard interface. We include this column to help the reader understand the discussion on it in the paper. The example list is made up based on a similar list we saw in real electronic medical record data at the University of Washington Medicine. For the feature value “2 of the number of ED visits related to asthma that the patient had in the prior 12 months,” the related rows in the list producing the feature value are marked in *italics*. HMC stands for Harborview Medical Center. UWMC stands for University of Washington Medical Center.

Visit date	Primary diagnosis	Visit type	Department	Provider	Facility
Dec-20-2020	Cough (R05)	Outpatient	HMC family medicine clinic	John Smith	HMC

Dec-18-2020	Dysphagia, unspecified (R13.10)	Outpatient	HMC family medicine clinic	David Wong	HMC
...
Oct-15-2020	Cystitis, unspecified without hematuria (N30.90)	Inpatient	UWMC 8SE	Leslie Hurdle	UWMC
Oct-12-2020	<i>Viral infection, unspecified (B34.9)</i>	<i>Emergency</i>	<i>HMC HEDUCC</i>	<i>Patricia Sward</i>	<i>HMC</i>
Oct-09-2020	Dizziness and giddiness (R42)	Outpatient	HMC family medicine clinic	Eve Johnson	HMC
...
Feb-11-2020	Post-traumatic stress disorder, unspecified (F43.10)	Outpatient	HMC psychotherapy clinic	Amy Jiang	HMC
Feb-08-2020	<i>Syncope and collapse (R55)</i>	<i>Emergency</i>	<i>HMC HEDUCC</i>	<i>Peter Shavlik</i>	<i>HMC</i>
Feb-03-2020	Headache, unspecified (R51.9)	Outpatient	HMC family medicine clinic	Jude Lake	HMC
...

Table 2. For the example list shown in Table 1 and the feature value “2 of the number of ED visits related to asthma that the patient had in the prior 12 months,” the parts that the user of the automated explaining function wants to see in the related raw data producing the feature value. HMC stands for Harborview Medical Center.

Visit date	Primary diagnosis	Department	Provider	Facility
Oct-12-2020	Viral infection, unspecified (B34.9)	HMC HEDUCC	Patricia Sward	HMC
Feb-08-2020	Syncope and collapse (R55)	HMC HEDUCC	Peter Shavlik	HMC

For instance, in the rule-style explanation shown above, the first item on its left hand side is the feature value “2 of the number of ED visits related to asthma that the patient had in the prior 12 months.” Asthma may or may not be the primary diagnosis of either of these 2 visits. For this feature value, the user of the automated explaining function wants to see the relevant parts of these 2 visits (visit date, primary diagnosis, department handling the visit, admitting provider, facility where the visit occurred) in reverse chronological order (see Table 2), like the way encounters are displayed on the standard interface of an electronic medical record system. The patterns embedded in these parts give additional information on the patient not shown by the feature value, such as the time between these 2 visits, how long ago these 2 visits occurred, the primary diagnoses of these 2 visits, and whether these 2 visits occurred at the same facility. However, finding these parts is non-trivial. As we have seen in real electronic medical record data at the University of Washington Medicine, Intermountain Healthcare, and Kaiser Permanente Southern California, the patient could have over 100 encounters in the prior 12 months. Only a few of these encounters are ED visits, and even fewer of them are ED visits related to asthma. To find the ED visits of the patient in the prior 12 months, the user would need some manual effort even if aided by the search function for the electronic medical record system. To figure out which of these visits are related to asthma, a task with which the search function often cannot provide much help, the user would need much more manual effort.

In practice, numerous possible features computed by various aggregation functions on all kinds of longitudinal attributes in the electronic medical records could be used for predictive modeling and automatic explanation. Examples of such features include whether the most recent asthma diagnosis of the patient is a primary diagnosis, the patient’s average respiratory rate recorded in the prior 12 months, the total number of distinct asthma medications ordered for the patient in the prior 12 months, the total number of units of asthma relievers that were ordered for the patient in the prior 12 months and were neither systemic corticosteroids nor short-acting beta-2 agonists, the number of distinct asthma medication prescribers of the patient in the prior 12 months, and the number of no shows by the patient in the prior 12 months [13,14]. Most of the possible features are unanticipated by the developers of the search function for the electronic medical record system beforehand. The search function supports only a few fixed types of search. For only a small portion of possible features, the search function can aid drilling through the raw data that produce a given feature value.

This creates a problem for the widespread adoption of our automatic explanation method for machine learning predictions. Frequently, our method gives multiple rule-style explanations for a patient predicted to be at high risk of incurring a poor outcome [11,12]. The user of the automated explaining function is typically a busy clinician having no time to do laborious manual drill-through regularly. However, to better understand the patient’s situation and to make better clinical decisions, the user often wants to drill through multiple feature values of the patient appearing in the explanations. If done manually, this is a challenging task. A patient often has extensive records with numerous variables and hundreds of pages of content accumulated over a long period of time [15]. Also, the relevant raw data producing the feature values are frequently scattered in several places in the electronic medical record system.

This paper makes 2 contributions towards solving this problem:

- 1) We articulate this problem for the first time in the literature. This is done in the “Introduction” section.

- 2) To address this problem, we outline an automated lineage tracing approach that adds automated drill-through capability to the automated explaining function, providing a roadmap for future research. This is done in the “An outline of our proposed automated lineage tracing approach for explaining machine learning predictions for clinical decision support” and the “Directions for future research” sections.

By offering the automated drill-through capability, we intend to help the user of the automated explaining function save time, better understand the patient’s situation, and make better clinical decisions. To let the reader have a concrete feeling, our discussion in this paper focuses on structured electronic medical record data, a specific method commonly used to build clinical machine learning predictive models, and our automatic explanation method for machine learning predictions [11,12]. Nevertheless, our automated lineage tracing approach is not limited to them. Instead, when automatically explaining machine learning predictions and after appropriate extension, the principle of our approach can be applied to facilitate drilling through any feature value computed by an aggregation function on longitudinal structured data, regardless of whether the data come from electronic medical records, whether the feature is specified by a human expert or semi-automatically extracted from longitudinal data using the method outlined in our paper [16], which method is used to build the machine learning predictive model, and which automatic explanation method is used.

Running Example

To illustrate our approach, we use a running example throughout this paper: automatically explaining the predictions of future ED visits or inpatient stays for asthma on individual asthma patients. Our prior papers [12-14,17-19] detail this use case and the features used to make predictions in it.

Base tables

Below are the schemas of 5 tables in a relational database used in the running example:

encounter (encounter_id, patient_id, encounter_type, admit_time, department, admitting_provider, facility, ...),
 diagnosis (encounter_id, dx_sequence_number, ICD_version, diagnosis_code, ...),
 diagnosis_code_master (ICD_version, diagnosis_code, dx_code_description, ...),
 ordered_medication (order_id, medication_id, patient_id, encounter_id, ordering_time, start_time, end_time, quantity, dose_unit, refills, ordering_provider, ...),
 medication_master (medication_id, name, ...).

The underlined fields mark the key of each table. The *encounter* table includes 1 row per encounter listing its information. The *diagnosis* table includes 1 row per diagnosis code of an encounter. Primary diagnoses are signified by *dx_sequence_number=1*. The *diagnosis_code_master* table includes 1 row per unique diagnosis code giving its description. The *ordered_medication* table includes 1 row per medication appearing in a medication order. The *medication_master* table includes 1 row per unique medication listing its information.

Intermediate result tables

Besides the above 5 base tables, we also use 4 intermediate result tables computed on the new data in the running example: *enc_features_1*, *enc_features_2*, *enc_features_3*, and *med_features_1*. The trained machine learning predictive model is applied to the new data to make predictions on individual patients.

The intermediate result table *enc_features_1* contains 3 temporal features on encounters: the number of ED visits, the number of inpatient stays, and the number of outpatient visits that the patient had in the prior 12 months. Let *today_date* denote today’s date. *enc_features_1* is computed from the *encounter* base table using the following Structured Query Language (SQL) query:

```
Q1: create table enc_features_1 as
select patient_id,
       sum(case when encounter_type = 'emergency' then 1 else 0 end) as count_ED_visits,
       sum(case when encounter_type = 'outpatient' then 1 else 0 end) as count_outpatient_visits,
       sum(case when encounter_type = 'inpatient' then 1 else 0 end) as count_inpatient_stays
from encounter
where admit_time between today_date - 365 and today_date
group by patient_id;
```

The intermediate result table *enc_features_2* contains 1 temporal feature on encounters: the number of outpatient visits with a primary diagnosis of asthma that the patient had in the prior 12 months. Recall that the International Classification of Diseases, Tenth Revision (ICD-10) diagnosis codes of asthma are J45.x. *enc_features_2* is computed by joining the *encounter* and *diagnosis* base tables using the following SQL query:

```

Q2: create table enc_features_2 as
select e.patient_id,
       count(*) as count_outpatient_visits_for_asthma
from encounter e, diagnosis d
where e.encounter_id = d.encounter_id
      and e.admit_time between today_date - 365 and today_date
      and e.encounter_type = 'outpatient'
      and d.ICD_version = 'ICD10'
      and d.diagnosis_code like 'J45.%'
      and d.dx_sequence_number = 1
      -- primary diagnosis
group by e.patient_id;

```

The intermediate result table *enc_features_3* contains 2 temporal features on encounters: the number of ED visits related to asthma and the number of inpatient stays related to asthma that the patient had in the prior 12 months. *enc_features_3* is computed by joining the *encounter* and *diagnosis* base tables using the following SQL query:

```

Q3: create table enc_features_3 as
select e.patient_id,
       sum(case when e.encounter_type = 'emergency' then 1 else 0 end) as count_ED_visits_related_to_asthma,
       sum(case when e.encounter_type = 'inpatient' then 1 else 0 end) as count_inpatient_stays_related_to_asthma
from encounter e,
     (select distinct encounter_id
      from diagnosis
      where ICD_version = 'ICD10'
        and diagnosis_code like 'J45.%'
     ) e_id
where e.encounter_id = e_id.encounter_id
      and e.admit_time between today_date - 365 and today_date
group by e.patient_id;

```

The intermediate result table *med_features_1* contains 2 temporal features on medications: the total number of medications and the total number of distinct medications ordered for the patient in the prior 12 months. *med_features_1* is computed from the *ordered_medication* base table using the following SQL query:

```

Q4: create table med_features_1 as
select patient_id,
       count(*) as count_medications_ordered,
       count(distinct medication_id) as count_distinct_medications_ordered
from ordered_medication
where ordering_time between today_date - 365 and today_date
group by patient_id;

```

Relational algebra operators

This paper uses the following relational algebra operators with the bag semantics unless otherwise specified: join \bowtie , left semijoin \ltimes , selection σ , projection π , duplicate elimination δ , and grouping γ [20]. Commercial database management systems implement relations using the bag semantics.

Review of a Typical Method to Build a Clinical Machine Learning Predictive Model and Our Automated Method to Explain the Model's Predictions

In this section, we review a typical method to build a machine learning predictive model on structured electronic medical record data, as well as our automated method to explain the model's predictions. In the next section, we outline our automated lineage tracing approach based on these 2 methods.

A healthcare system usually has an enterprise data warehouse. It stores in a relational database a copy of the structured electronic medical record data of the healthcare system, often after some transformations such as pivoting [21,22] and denormalization to facilitate data analysis. For predictive modeling with automated explanation, the overall workflow is to execute database SQL queries to extract features from the electronic medical record data, to build a machine learning predictive model on the training data, to apply the model on new data to make predictions on individual patients, and then to use our automated method to explain the predictions. In the following sections, we describe each of these steps sequentially.

Extracting features from the electronic medical record data and building the clinical machine learning predictive model

The structured electronic medical record data contain both static attributes (e.g., gender) and longitudinal attributes (e.g., encounters and diagnoses). Most attributes are longitudinal. As Figure 1 shows, we perform the following operations on the training data:

- 1) We compute static features from the static attribute values and store the results in 1 or more intermediate result tables. Typically, each of these intermediate result tables is computed by running a select-project-join SQL query on 1 or more base tables.
- 2) By aggregating longitudinal attribute values and sometimes also using some static attribute values, we compute the patient cohort of interest in the training data and store the result in 1 intermediate result table. This is typically done by running a complex SQL query on several base tables. An example patient cohort is the set of all asthma patients who visited any of the facilities of the healthcare system during a specific time period.
- 3) By aggregating longitudinal attribute values, we compute temporal features and the outcome variable and store them in 1 or more intermediate result tables. Typically, each of these intermediate result tables is computed by running a select-project-join-aggregate SQL query on 1 or more base tables. For example, 1 intermediate result table is similar to *enc_features_1* and contains multiple temporal features on encounters computed from the *encounter* base table. A second intermediate result table is similar to *enc_features_2* and contains multiple temporal features on encounters computed by joining the *encounter* and *diagnosis* base tables. A third intermediate result table contains multiple temporal features on medications computed by joining the *ordered_medication* and *medication_master* base tables, such as the total number of distinct asthma medications and the total number of units of asthma medications ordered for the patient in the prior 12 months. As Figure 2 shows, the logical query plan for a select-project-join-aggregate query includes 1 or more select-project-join-aggregate segments [23]. Each segment has a grouping or duplicate elimination operator at its end following a bunch of join, selection, and projection operators.

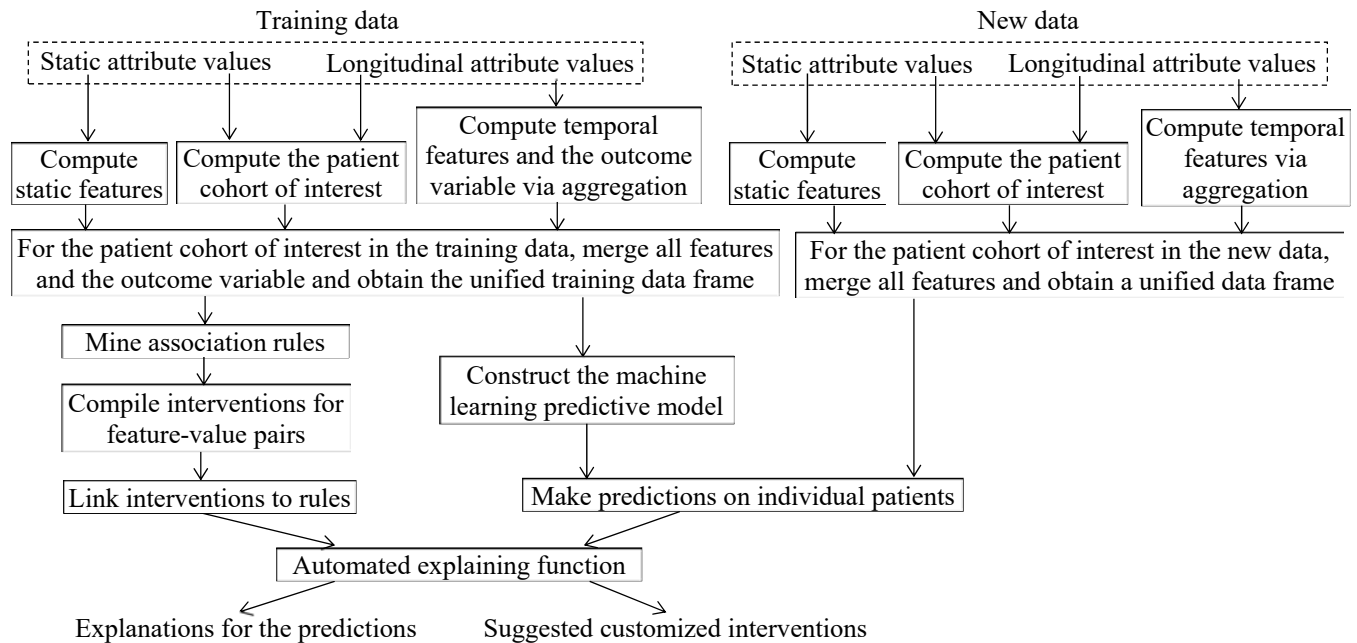


Figure 1. The flow chart of building a clinical machine learning predictive model on the training data, making predictions on the new data, and using our automated method to explain the model's predictions.

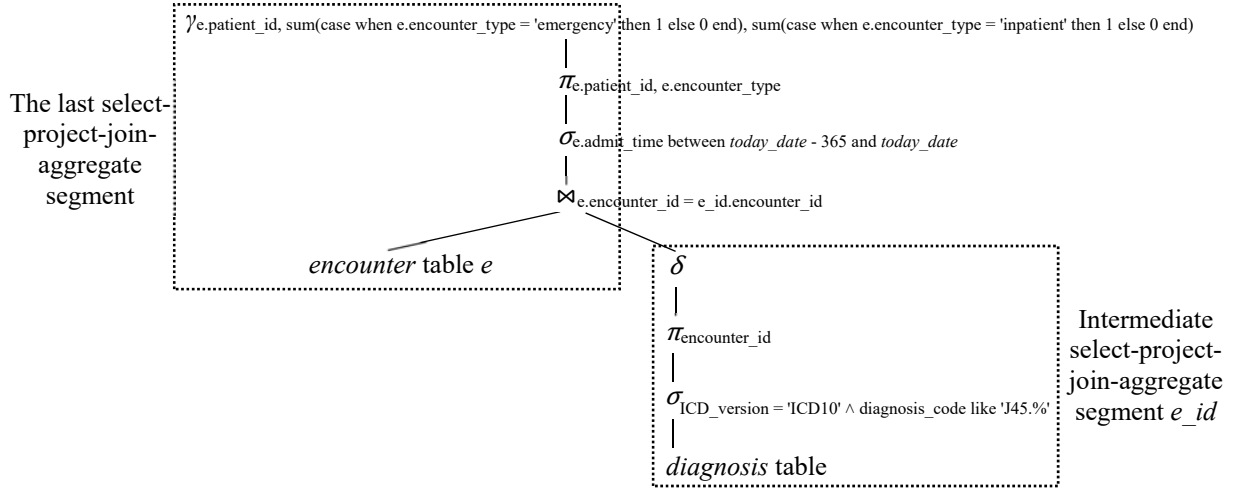


Figure 2. A logical query plan for the select-project-join-aggregate query Q_3 given in the “Intermediate result tables” section.

By joining the intermediate result tables containing the patient cohort of interest, the static and temporal features, and the outcome variable in the training data, we obtain a table containing the unified training data frame. For the patient cohort of interest, this table includes 1 column for the outcome variable and a separate column for each feature. Then a machine learning predictive model is trained on this table.

Applying the machine learning predictive model to the new data to make predictions on individual patients

As Figure 3 shows, using a procedure similar to the above, we compute the patient cohort of interest and the static and temporal features in the new data and store the results in several intermediate result tables. By joining these tables, we obtain a table containing the unified data frame for the new data. For the patient cohort of interest, this table includes a separate column for each feature. We then apply the machine learning predictive model to this table to make predictions on individual patients.

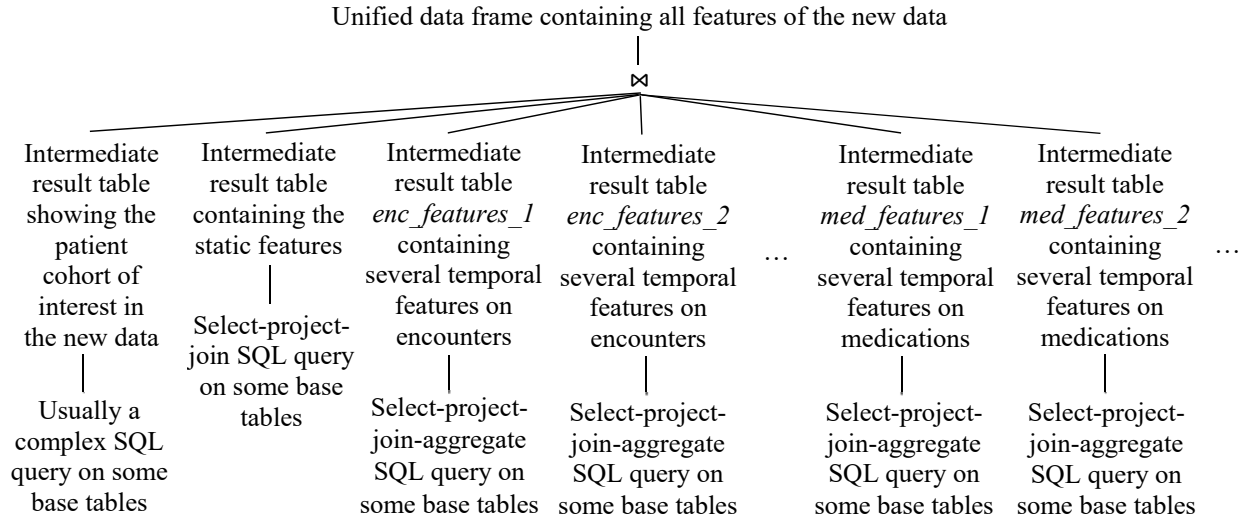


Figure 3. The high-level logical query plan for computing the unified data frame that contains all features of the new data.

Automatically explaining the machine learning model’s predictions

At the same time of building the clinical machine learning predictive model, we use the training data to create the knowledge base of the automated explaining function. We do automated discretization [24,25] to convert continuous features to categorical features. Then we mine class-based association rules [24,26] from the unified training data frame. Each rule delineates the association between a feature value pattern and a poor outcome value c , and is of the form

$$i_1 \text{ AND } i_2 \text{ AND } \dots \text{ AND } i_t \rightarrow c.$$

The rule shows that a patient satisfying $i_1, i_2, \dots,$ and i_t tends to have outcome value c . The values of t and c can change across rules. Each item i_k ($1 \leq k \leq t$) is a feature-value pair showing that a feature has a specific value or a value within a specific range. One example item of the former is that the patient had 2 ED visits related to asthma in the prior 12 months. One example item of the latter is that the patient's average respiratory rate recorded in the prior 12 months is >25 and ≤ 28 breaths per minute. An example rule containing both items is given in the Introduction.

For each feature-value pair item used to create association rules, we pre-compile 0 or more interventions. The interventions pre-compiled for any item on a rule's left hand side are automatically linked to the rule.

At prediction time, to avoid reducing the machine learning predictive model's performance measures, we use the model's predictions with no change. The mined association rules are used to explain these predictions rather than to make predictions. More specifically, for each patient the model predicts to have a poor outcome value, we find and display the rules with this value on their right hand sides and whose left hand sides are fulfilled by the patient. Each rule offers 1 explanation for the prediction. The interventions linked to the rule are displayed next to it as the suggested candidate interventions.

Our automatic explanation method for machine learning predictions has been successfully applied to multiple clinical predictive modeling problems [11,12,27,28]. It has several advantages. Among all of the automatic explanation methods for machine learning predictions in the literature [29,30], our method is the only one that can automatically suggest customized interventions. The rule-style explanations given by our method are easier to comprehend than the non-rule-style explanations given by many other methods. Unlike many other automatic explanation methods that either lower the machine learning predictive model's performance measures or work for only a specific machine learning algorithm, our method works for any machine learning algorithm on tabular data without lowering the model's performance measures. Unlike several other methods that use rules computed at prediction time to offer explanations [31,32], our method uses rules mined before prediction time to offer explanations. This is essential for our method to automatically suggest customized interventions at prediction time.

Review of the existing automated lineage tracing techniques

In this section, we review the existing automated lineage tracing techniques. We first give an overview of such techniques developed in various fields. Then we review a specific set of automated lineage tracing techniques most closely related to this work.

An overview of the existing automated lineage tracing techniques

The lineage, or provenance, of a given data item i refers to the source data items producing i and how i was derived [33]. The former is called where-lineage. The latter is called how-lineage. Each type of lineage can be at either the schema level or the instance level. An example of where-lineage at the schema level is the set of base tables producing a specific materialized view. An example of where-lineage at the instance level is the set of tuples in the base tables producing a given temporal feature value in a materialized view. Lineage information can be computed in either an eager way or a lazy way. In the former case, lineage information is computed and stored at the same time of producing the output data. In the latter case, lineage information is computed when needed. This paper focuses on where-lineage that is at the instance level and computed in a lazy way.

Ikeda *et al.* surveyed existing lineage tracing techniques in databases [33,34], e-science [35], and scientific data processing [36]. Among all of the lineage tracing techniques in the literature, the techniques Cui *et al.* [23,37] developed are the most closely related to this work. These techniques are used to trace the lineage of a tuple in a materialized view [38] defined by a select-project-join-aggregate query in a relational database. Cui *et al.* [39,40] described lineage tracing techniques for warehouse data computed via a directed acyclic graph of transformations, some of which could involve complex procedural code. Zhang *et al.* [41] described lineage tracing techniques for data computed by arbitrary functions. In general, the more flexibility is allowed on the transformations or functions, the less efficiently lineage can be traced [39].

In big data systems, Ikeda *et al.* [42,43] described lineage tracing techniques for data computed via a directed acyclic graph of map and reduce functions [44]. Amsterdamer *et al.* [45] described lineage tracing techniques for data computed using Pig Latin [46].

In scientific data processing, lineage tracing is often done on curated databases, which contain scientific data copied from other databases [36,47].

Schelter *et al.* [48] described a method to trace the schema-level lineage of the data sets, features, models, and predictions produced in machine learning experiments.

Review of Cui *et al.*'s automated lineage tracing techniques [23,37] for relational databases

To automatically trace the lineage of a tuple t in a materialized view [38] defined by a select-project-join-aggregate query, Cui *et al.* [23,37] proceed as follows. First, we transform the materialized view's definition query into a canonical form of the logical query plan. As Figure 2 shows, the canonical form includes 1 or more select-project-join-aggregate segments. Each segment has 0 or 1 join operator, 0 or 1 selection operator, 0 or 1 projection operator, and a grouping or duplicate elimination

operator in this particular order. Second, we create an intermediate materialized view for each intermediate select-project-join-aggregate segment of the canonical form. The root node of such a segment is not the root node of the canonical form. Third, we recursively trace through the hierarchy of intermediate materialized views in a top-down way. At each level of the hierarchy, we use the lineage tracing query for a 1-level select-project-join-aggregate materialized view to compute the current traced tuples' lineage with respect to each base table and each materialized view at the next lower level. For a 1-level select-project-join-aggregate materialized view $MV = \gamma(\pi_A(\sigma_C(R_1 \bowtie R_2 \bowtie \dots \bowtie R_n)))$, the lineage of a tuple set $T \subseteq MV$ with respect to the base table or the materialized view R_i ($1 \leq i \leq n$) is $\pi_{R_i}(\sigma_C(R_1 \bowtie R_2 \bowtie \dots \bowtie R_n) \bowtie T)$. Here, the projection operator π on R_i has the set semantics, making each selected tuple in R_i appear only once. Also, all attributes of R_i appear in the projection operator and subsequently in the lineage traced on R_i . The final traced lineage of tuple t includes the lineage traced on every base table appearing in the canonical form.

We use an example to illustrate Cui *et al.*'s [23,37] automated lineage tracing techniques. If we replace “create table `enc_features_3`” by “create materialized view `enc_features_3_view`” in query Q_3 given in the “Intermediate result tables” section, we obtain a query Q_{3_v} defining a materialized view `enc_features_3_view`. To trace the lineage of a tuple t in `enc_features_3_view` whose `patient_id` is `asthma_patient_id`, we proceed as follows.

First, we obtain the canonical form of the logical query plan for query Q_{3_v} . The canonical form is the same as the logical query plan for query Q_3 shown in Figure 2.

Second, we create an intermediate materialized view `asthma_encounter_id` for the intermediate select-project-join-aggregate segment `e_id` shown in Figure 2:

```
Q5: create materialized view asthma_encounter_id as
    select distinct encounter_id
    from diagnosis
    where ICD_version = 'ICD10'
    and diagnosis_code like 'J45.%';
```

Figure 4 shows the resulting hierarchy of intermediate materialized views, with the materialized view `enc_features_3_view` at the top and the `encounter` and `diagnosis` base tables at the bottom.

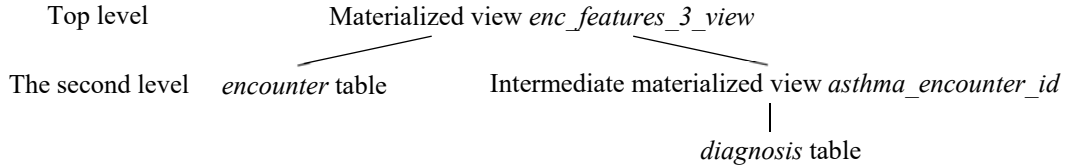


Figure 4. The hierarchy of intermediate materialized views matching the canonical form of the logical query plan for the definition query of the materialized view `enc_features_3_view`.

Third, at the top level of the hierarchy of intermediate materialized views, we compute the lineage of tuple t with respect to the `encounter` base table using the following SQL query:

```
Q6: select e.*
    from encounter e
        inner join asthma_encounter_id e_id
            on e.encounter_id = e_id.encounter_id
    where e.admit_time between today_date - 365 and today_date
        and e.patient_id = asthma_patient_id;
```

Using the following SQL query, we compute the lineage of tuple t with respect to the intermediate materialized view `asthma_encounter_id` and store the results in a temporary table `temp`:

```
Q7: create temporary table temp as
    select e_id.*
    from encounter e
        inner join asthma_encounter_id e_id
            on e.encounter_id = e_id.encounter_id
    where e.admit_time between today_date - 365 and today_date
```

and e.patient_id = asthma_patient_id;

Fourth, at the second level of the hierarchy of intermediate materialized views, we compute the lineage of the tuples in the temporary table *temp* with respect to the *diagnosis* base table using the following SQL query:

```

Q8: select d.*
      from diagnosis d
      inner join temp t
            on d.encounter_id = t.encounter_id
      where d.ICD_version = 'ICD10'
            and d.diagnosis_code like 'J45.%';

```

The final traced lineage of tuple *t* includes both the results of query *Q₆* and the results of query *Q₈*.

An Outline of Our Proposed Automated Lineage Tracing Approach for Explaining Machine Learning Predictions for Clinical Decision Support

In this section, we outline an automated lineage tracing approach that adds automated drill-through capability to the automated explaining function. Our presentation includes 4 subsections. In the first subsection, we give an overview of the lineage tracing component of the automated explaining function. In the second subsection, we point out the unique requirements on automated lineage tracing for automatically explaining machine learning predictions for clinical decision support. In the third subsection, we outline our proposed automated lineage tracing techniques fulfilling these requirements. In the fourth subsection, we present some considerations for future computer coding implementation of our proposed lineage tracing approach.

An overview of the lineage tracing component of the automated explaining function

At association rule mining time, we already know all feature-value pair items used to create association rules, as well as which items involve temporal features computed by aggregation functions on the raw data. For each item that is related to a temporal feature of a patient and on the left hand side of a rule, we add a hyperlink to the item in the rule. In addition, we write a parameterized stored procedure for the item in the database to retrieve lineage information. The stored procedure typically has 2 parameters: the *patient_id* of the patient being examined and the endpoint of the temporal aggregation period, such as today. When the stored procedure is run for the first time, an execution plan is generated. All subsequent runs will use the same execution plan to avoid runtime query optimization overhead.

At automatic explanation time, we allow the user of the automated explaining function to do lineage tracing for any item that is on the left hand side of a rule-style explanation and related to a temporal feature value. When the user clicks the item’s hyperlink, the stored procedure pre-written for the item is invoked to retrieve some pre-specified parts of the related raw data producing the feature value. Except for the cases with 2 specific aggregation functions described later in the paper, the retrieved data instances are always displayed on a page in reverse chronological order like that in the electronic medical records.

The unique requirements on automated lineage tracing for automatically explaining machine learning predictions for clinical decision support

Typically, the user of the automated explaining function is a clinician. To fit the user’s busy schedule and to aid timely decision making, the user wants the lineage tracing process for a temporal feature value to be finished quickly, preferably within 1 second. This goal is partially fulfilled by the existing lineage tracing techniques [23,37], whereas the realized lineage tracing speed can be further improved. In addition, the retrieved lineage information should be easy to scan and include the most essential content needed to facilitate decision making. This enables the user to quickly gain useful insights from the information, ideally within 1 or a few seconds. As summarized in Table 3, that goal translates to 5 unique requirements on automated lineage tracing that are unmet by the existing lineage tracing techniques.

Table 3. The 5 unique requirements on automated lineage tracing for automatically explaining machine learning predictions for clinical decision support.

Requirement	Reason for posing the requirement
#1: Retrieving only a small set of attributes	To prevent the user from being overwhelmed by many non-essential or irrelevant attributes
#2: Adding some essential attributes that do not directly produce the feature value	To make the retrieved lineage information include the most essential content

#3: Sorting the retrieved lineage information in an appropriate order	To make the retrieved lineage information easy to scan
#4: Computing the lineage information based on the semantic meaning of the feature	To avoid including irrelevant or non-essential source tuples in the retrieved lineage information
#5: Performing no lineage tracing for any healthcare system feature value computed by an aggregation function	To avoid including irrelevant data in the retrieved lineage information

Requirement 1: When tracing the lineage of a temporal feature value, we should retrieve from the base tables only a small set of attributes specific to the temporal feature rather than the many attributes involved in deriving all of the features used for automated explanation.

We pose this requirement to prevent the user of the automated explaining function from being overwhelmed by many non-essential or irrelevant attributes. To aid automatic explanation, we want to allow tracing the lineage of a temporal feature value in the form of a small set of attributes specific to the temporal feature (see Table 2 for an example). This cannot be well done using Cui *et al.*'s lineage tracing techniques [23,37]. These techniques were developed to trace the lineage of a tuple including all of its attribute values in a select-project-join-aggregate materialized view in a relational database. If the retrieved lineage information ever touches a tuple in a base table, all attribute values of the tuple are included in this information. For automatic explanation, both factors would cause the retrieved lineage information to have an excessive volume, overwhelming the user of the automated explaining function.

To see this, we review the process of making predictions with automatic explanations. Usually, many features are used to make predictions and to automatically explain them. All of the items on the left hand side of a rule-style explanation come from the same tuple in the unified data frame, which contains all features of the new data. As Figure 3 shows, this unified data frame is obtained by joining many intermediate result tables. Each of them falls into 1 of 3 categories: 1) a table containing the patient cohort of interest in the new data, 2) a table containing 1 or more static features, and 3) a table containing 1 or more temporal features. Each hyperlinked item on the left hand side of a rule-style explanation comes from exactly 1 intermediate result table in the third category.

When the user of the automated explaining function clicks the hyperlink for an item on the left hand side of a rule-style explanation, one could use Cui *et al.*'s techniques [23,37] to trace the lineage of the tuple in the unified data frame, from which the item comes. For each intermediate result table mentioned above and each base table used to create it, the retrieved lineage information contains some tuples from the base table including all of their attribute values. Most of the retrieved lineage information is unnecessary for automatic explanation for 3 reasons.

Reason 1

The retrieved lineage information often includes thousands of tuples from several dozen base tables. Most of these base tables are used to compute the other feature values in the tuple in the unified data frame that are unrelated to the item, and include no information that can help the user of the automated explaining function gain useful insights related to the item. In fact, to obtain the lineage information of the item essential for automatic explanation, we need to only trace through the intermediate result table related to the item solely for the item and to examine the base tables used to create this table. The features in this table that are unrelated to the item can be ignored. There is also no need to trace through the intermediate result tables containing the features unrelated to the item. Moreover, at automatic explanation time, we know the *patient_id* of the patient linked to the item. The user usually does not need to know why this patient is in the patient cohort of interest in the new data. Thus, there is no need to trace through the intermediate result table showing the patient cohort.

Reason 2

A base table often has many attributes, only a few of which are essential for the user of the automated explaining function to gain useful insights related to the item. For instance, the *encounter* table often has >100 attributes. The lineage information shown in Table 2 covers only 4 of them: *admit_time* transformed to the date format, *department*, *admitting_provider*, and *facility*.

Reason 3

Certain items are each computed using several base tables and intermediate query results. For the user of the automated explaining function to gain useful insights related to the item, only the attributes and tuples of some of these base tables are essential. Alternatively, none or only some of these intermediate query results need to be traced through.

For example, in query Q_2 given in the "Intermediate result tables" section, both the *encounter* and *diagnosis* base tables are used to compute the feature "the number of outpatient visits with a primary diagnosis of asthma that the patient had in the prior 12 months." For a value of this feature, we need to use the information in the *diagnosis* table to find the related tuples in the *encounter* table. Nevertheless, the user would expect each encounter shown in the retrieved lineage information to be an

outpatient visit with a primary diagnosis of asthma. Thus, there is no need to include any attribute or tuple from the *diagnosis* table in the retrieved lineage information, e.g., to give the primary diagnosis of each encounter included in that information.

As a second example, in query Q_3 given in the “Intermediate result tables” section, both the *encounter* base table and the intermediate query result e_id are used to compute the feature “the number of ED visits related to asthma that the patient had in the prior 12 months.” For a value of this feature, the user of the automated explaining function would expect each encounter shown in the retrieved lineage information to be an ED visit related to asthma, like that shown in Table 2. Thus, there is no need to trace through e_id and to obtain the corresponding tuples in the *diagnosis* table showing that each encounter included in the retrieved lineage information has an asthma diagnosis code.

Requirement 2: For certain temporal features, when acquiring the lineage of a feature value, we should not use just the related raw data that directly produce the feature value. Instead, we need to add to them some related attributes in the base tables, which are specific to the temporal feature and do not directly produce the feature value.

We pose this requirement to make the retrieved lineage information include the most essential content needed to facilitate decision making. For example, as query Q_1 given in the “Intermediate result tables” section shows, the feature “the number of ED visits that the patient had in the prior 12 months” is computed solely from the *encounter* base table. For a value of this feature, we want the retrieved lineage information to be similar to that shown in Table 2 and include a primary diagnosis column. This column is computed using the *diagnosis* and *diagnosis_code_master* base tables unused in Q_1 , and is formed by concatenating the *diagnosis_code* and *dx_code_description* columns of the *diagnosis_code_master* base table. The cases for many other temporal features on encounters are similar.

Requirement 3: When presenting the lineage information, the related raw data retrieved for a temporal feature value should be sorted in an order specific to the temporal feature.

We pose this requirement to make the retrieved lineage information easy to scan. Usually, we want the data instances in the retrieved lineage information to be displayed in reverse chronological order like that in the electronic medical records. However, there are 2 exceptions. First, when the temporal feature is the maximum value of an attribute of a given patient, we want the related raw data retrieved for a feature value to be displayed in descending order of the attribute value. For example, for the feature “the highest systolic blood pressure of the patient in the prior 12 months,” we want the lineage information retrieved for a feature value to contain the systolic blood pressures of the patient in the prior 12 months sorted in descending order. Second, when the temporal feature is the minimum value of an attribute of a given patient, we want the related raw data retrieved for a feature value to be displayed in ascending order of the attribute value. In either of the 2 cases, we could add a re-sort button to the retrieved lineage information on display. If the user of the automated explaining function clicks this button, the data instances in the retrieved lineage information are re-arranged in reverse chronological order for display.

Requirement 4: The lineage information of a temporal feature value should be computed based on the semantic meaning of the feature rather than solely on the literal writing of the SQL query used to compute the feature.

We pose this requirement to avoid including irrelevant or non-essential source tuples in the retrieved lineage information. For a select-project-join-aggregate materialized view containing 1 or more temporal features, Cui *et al.* [23,37] compute the lineage of a tuple in it based solely on the literal SQL query used to define it. In certain cases, this literal approach is suboptimal for automatic explanation. Instead, we should consider the semantic meanings of the temporal features during lineage tracing. In the following, we describe 2 such cases, each presented as a sub-requirement.

Sub-requirement 4.1: When the temporal feature is the sum of a variable computed by a case statement in SQL including multiple conditions and some of them return 0, we should retrieve only the lineage information related to the other conditions.

In SQL, such a temporal feature is written in the form of

```
sum(case when condition1 then result1
      when condition2 then result2
      ...
      when conditionn then resultn
      else resultn+1
end).
```

As an example of this sub-requirement, for the feature “the number of ED visits that the patient had in the prior 12 months,” the lineage information retrieved for a value of the feature should be the ED visits that the patient had in the prior 12 months, regardless of whether the feature is computed using SQL query Q_9 or Q_{10} below:

```
 $Q_9$ : select patient_id,
       sum(case when encounter_type = 'emergency' then 1 else 0 end) as count_ED_visits
from encounter
where admit_time between today_date - 365 and today_date
group by patient_id;
```

```
 $Q_{10}$ : select patient_id,
         sum(1) as count_ED_visits
         -- sum(1) can be replaced by count(*)
from encounter
where admit_time between today_date - 365 and today_date
      and encounter_type = 'emergency'
group by patient_id;
```

The differences between Q_9 and Q_{10} are highlighted in italics in Q_{10} . If the feature is computed using Q_9 , Cui *et al.*'s techniques [23,37] would retrieve all encounters of the patient in the prior 12 months as the lineage information. This could easily overwhelm the user of the automated explaining function, as usually most of these encounters are not ED visits.

Sub-requirement 4.2: When the temporal feature is the total number of distinct items, the retrieved lineage information should include only 1 representative data instance for each distinct item.

For example, query Q_4 given in the “Intermediate result tables” section computes the feature “the total number of distinct medications ordered for the patient in the prior 12 months.” For a value of this feature, Cui *et al.*'s techniques [23,37] would retrieve all medications ordered for the patient in the prior 12 months as the lineage information. This information is often overwhelming and not succinct enough for the user of the automated explaining function to quickly find the distinct medications ordered for the patient in the prior 12 months, as the same medication could be ordered for the patient multiple times in a year. To avoid this problem, we could retrieve only the most recent order of each distinct medication ordered for the patient in the prior 12 months as the lineage information. For the user, these distinct medications typically provide enough insight into the patient's status related to the feature value.

Requirement 5: We do not trace the lineage of any healthcare system feature value computed by an aggregation function.

We pose this requirement to avoid including irrelevant data in the retrieved lineage information. Like temporal features of a patient, certain healthcare system features [17-19], such as the number of asthma patients of the primary care provider of a patient, are computed by aggregation functions. These healthcare system features are each computed using multiple patients' information rather than solely the information of the patient being examined. Since other patients' detailed information does not help the user of the automated explaining function understand this patient's situation, we do not trace the lineage of any value of this feature, even if it appears on the left hand side of a rule-style explanation.

An outline of our proposed techniques to form the lineage tracing query that computes the lineage information

To perform automated lineage tracing for explaining machine learning predictions for clinical decision support, we modify Cui *et al.*'s lineage tracing techniques [23,37] to fulfill the requirements mentioned above. Even without giving any detail on the computer coding implementation and the performance evaluation results, Cui *et al.* [37] already took 49 pages to describe the details of their automated lineage tracing algorithm. Our case is more complex than Cui *et al.*'s case [37]. In our case, which attributes are most relevant and which source tuples are most essential for inclusion in the retrieved lineage information depend on both the concrete feature type and the clinical decision support application's need. In comparison, no such dependency exists in Cui *et al.*'s case [37]. Thus, we expect that, once fully worked out, our automated lineage tracing algorithm would be more sophisticated than Cui *et al.*'s algorithm [37]. In this viewpoint paper, we do not intend to enumerate all possible feature types and provide a detailed design or any computer coding implementation of our proposed automated lineage tracing approach. Rather, our goal is to describe the design approach for our proposed automated lineage tracing module and provide a roadmap for future research. We achieve this goal by outlining the main steps of forming the lineage tracing query, giving 4 example temporal features, and illustrating at a high level how to form the lineage tracing query for each of these 4 features.

An overview of the lineage tracing query formation process

Usually, each intermediate result table shown in Figure 3 has a *patient_id* column. It is used as the join column in the join operation to produce the unified data frame containing all features of the new data. As explained in “Reason 1” of the “Requirement 1” section, to obtain the lineage information of a temporal feature value, we need to only trace through the intermediate result table containing this value solely for this value. This intermediate result table is usually computed from some base tables by using a select-project-join-aggregate SQL query S_0 . To form the lineage tracing query for a temporal feature value of a patient in the intermediate result table, we proceed in 4 steps. First, we remove the other temporal features, if any, from S_0 and obtain a simplified query S_1 . Second, if applicable, we transform S_1 to query S_2 to fulfill Sub-requirement 4.1. Third, we modify Cui *et al.*'s techniques [23,37] to address Reasons 2 and 3 given in the “Requirement 1” section. Using the modified techniques, we form a preliminary lineage tracing query S_3 based on S_2 and the patient's *patient_id*. Fourth, we transform S_3 to fulfill Requirements 2 and 3 and Sub-requirement 4.2 and obtain the final lineage tracing query.

In the following, we use 4 examples to illustrate at a high level how to form the lineage tracing query. In each example, the user of the automated explaining function is examining an asthma patient whose identifier is *asthma_patient_id*, and wants to drill through a temporal feature value of this patient. We outline the main steps of forming the lineage tracing query for the feature value without giving the detailed algorithm.

Example 1: The number of ED visits that the patient had in the prior 12 months

As defined by query Q_1 in the “Intermediate result tables” section, the intermediate result table *enc_features_1* contains 3 temporal features. One of them is the number of ED visits that the patient had in the prior 12 months. To form the lineage tracing query for a value of this feature, we proceed as follows.

First, we remove the other 2 features from query Q_1 and obtain query Q_9 given in the “Sub-requirement 4.1” section.

Second, to fulfill Sub-requirement 4.1 on handling the sum of a variable computed by a case statement, we transform query Q_9 to query Q_{10} given in the “Sub-requirement 4.1” section.

Third, using Cui *et al.*'s lineage tracing techniques [23,37], we form a draft lineage tracing query Q_{11} based on Q_{10} and *asthma_patient_id*:

```
Q11: select *
from encounter
where admit_time between today_date - 365 and today_date
and encounter_type = 'emergency'
and patient_id = asthma_patient_id;
```

The differences between Q_{10} and Q_{11} are highlighted in italics in Q_{11} . To address Reason 2 given in the “Requirement 1” section and retrieve from the *encounter* table only its attributes essential for automatic explanation, we transform Q_{11} to the following preliminary lineage tracing query:

```
Q12: select cast(admit_time as date) as visit_date, department, admitting_provider, facility
from encounter
where admit_time between today_date - 365 and today_date
and encounter_type = 'emergency'
and patient_id = asthma_patient_id;
```

The differences between Q_{11} and Q_{12} are highlighted in italics in Q_{12} .

Fourth, to fulfill Requirement 2, we need to add a primary diagnosis column to the raw data that are retrieved by query Q_{12} and directly produce the feature value being examined. To fulfill Requirement 3, we need to sort the retrieved raw data in reverse chronological order. To meet both demands, we transform Q_{12} to the following final lineage tracing query:

```
Q13: select cast(e.admit_time as date) as visit_date,
case when m.diagnosis_code is null then null
else m.dx_code_description || '(' || m.diagnosis_code || ')'
end as primary_diagnosis,
e.department, e.admitting_provider, e.facility
from encounter e
left outer join diagnosis d
on e.encounter_id = d.encounter_id
```

```

left outer join diagnosis_code_master m
  on d.diagnosis_code = m.diagnosis_code
  and d.ICD_version = m.ICD_version
where e.admit_time between today_date - 365 and today_date
  and e.encounter_type = 'emergency'
  and e.patient_id = asthma_patient_id
  and d.dx_sequence_number = 1
  -- primary diagnosis
order by e.admit_time desc;

```

The differences between Q_{12} and Q_{13} are highlighted in italics in Q_{13} . `||` is the string concatenation operator in SQL.

Example 2: The number of outpatient visits with a primary diagnosis of asthma that the patient had in the prior 12 months

As defined by query Q_2 in the “Intermediate result tables” section, the intermediate result table *enc_features_2* contains the temporal feature “the number of outpatient visits with a primary diagnosis of asthma that the patient had in the prior 12 months.” To form the lineage tracing query for a value of this feature, we proceed as follows.

First, to address Reason 2 given in the “Requirement 1” section, we should include from the *encounter* table only its attributes essential for automatic explanation. To address Reason 3 given in the “Requirement 1” section, we should include no attribute or tuple from the *diagnosis* table in the retrieved lineage information. Using a modified version of Cui *et al.*'s lineage tracing techniques [23,37] that meets both demands, we form a preliminary lineage tracing query Q_{14} based on query Q_2 and *asthma_patient_id*:

```

Q14: select cast(e.admit_time as date) as visit_date,
  e.department, e.admitting_provider, e.facility
from encounter e, diagnosis d
where e.encounter_id = d.encounter_id
  and e.admit_time between today_date - 365 and today_date
  and e.encounter_type = 'outpatient'
  and d.ICD_version = 'ICD10'
  and d.diagnosis_code like 'J45.%'
  and d.dx_sequence_number = 1
  -- primary diagnosis
  and e.patient_id = asthma_patient_id;

```

The differences between Q_2 and Q_{14} are highlighted in italics in Q_{14} .

Second, to fulfill Requirement 3 of sorting the related raw data retrieved for the feature value in reverse chronological order, we transform query Q_{14} to the following final lineage tracing query:

```

Q15: select cast(e.admit_time as date) as visit_date,
  e.department, e.admitting_provider, e.facility
from encounter e, diagnosis d
where e.encounter_id = d.encounter_id
  and e.admit_time between today_date - 365 and today_date
  and e.encounter_type = 'outpatient'
  and d.ICD_version = 'ICD10'
  and d.diagnosis_code like 'J45.%'
  and d.dx_sequence_number = 1
  -- primary diagnosis
  and e.patient_id = asthma_patient_id
order by e.admit_time desc;

```

The differences between Q_{14} and Q_{15} are highlighted in italics in Q_{15} .

Example 3: The number of ED visits related to asthma that the patient had in the prior 12 months

As defined by query Q_3 in the “Intermediate result tables” section, the intermediate result table *enc_features_3* contains 2 temporal features. One of them is the number of ED visits related to asthma that the patient had in the prior 12 months. To form the lineage tracing query for a value of this feature, we proceed as follows.

First, we remove the other feature from query Q_3 and obtain the following simplified query:

```
Q16: select e.patient_id,
        sum(case when e.encounter_type = 'emergency' then 1 else 0 end) as count_ED_visits_related_to_asthma
from encounter e,
        (select distinct encounter_id
        from diagnosis
        where ICD_version = 'ICD10'
          and diagnosis_code like 'J45.%'
        ) e_id
where e.encounter_id = e_id.encounter_id
      and e.admit_time between today_date - 365 and today_date
group by e.patient_id;
```

Second, to fulfill Sub-requirement 4.1 on handling the sum of a variable computed by a case statement, we transform query Q_{16} to the following query:

```
Q17: select e.patient_id,
        sum(1) as count_ED_visits_related_to_asthma
from encounter e,
        (select distinct encounter_id
        from diagnosis
        where ICD_version = 'ICD10'
          and diagnosis_code like 'J45.%'
        ) e_id
where e.encounter_id = e_id.encounter_id
      and e.admit_time between today_date - 365 and today_date
      and e.encounter_type = 'emergency'
group by e.patient_id;
```

The differences between Q_{16} and Q_{17} are highlighted in italics in Q_{17} .

Third, to address Reason 2 given in the “Requirement 1” section, we should include from the *encounter* table only its attributes essential for automatic explanation. To address Reason 3 given in the “Requirement 1” section, we should not trace through the intermediate query result *e_id* and include any corresponding tuple in the *diagnosis* table in the retrieved lineage information. Using a modified version of Cui *et al.*'s lineage tracing techniques [23,37] that meets both demands, we form a preliminary lineage tracing query Q_{18} based on query Q_{17} and *asthma_patient_id*:

```
Q18: select cast(e.admit_time as date) as visit_date,
        e.department, e.admitting_provider, e.facility
from encounter e
      inner join (select distinct encounter_id
                from diagnosis
                where ICD_version = 'ICD10'
                  and diagnosis_code like 'J45.%'
                ) e_id
      on e.encounter_id = e_id.encounter_id
where e.admit_time between today_date - 365 and today_date
      and e.encounter_type = 'emergency'
      and e.patient_id = asthma_patient_id;
```

The differences between Q_{17} and Q_{18} are highlighted in italics in Q_{18} .

Applying Cui *et al.*'s lineage tracing techniques [23,37,49] to query Q_3 , we create a materialized view *asthma_encounter_id* defined by query Q_5 in the “Review of Cui *et al.*'s automated lineage tracing techniques [23,37] for relational databases” section. Using *asthma_encounter_id*, we rewrite the preliminary lineage tracing query Q_{18} as

```
 $Q_{19}$ : select cast(e.admit_time as date) as visit_date,
        e.department, e.admitting_provider, e.facility
from encounter e
    inner join asthma_encounter_id e_id
        on e.encounter_id = e_id.encounter_id
where e.admit_time between today_date - 365 and today_date
    and e.encounter_type = 'emergency'
    and e.patient_id = asthma_patient_id;
```

The differences between Q_{18} and Q_{19} are highlighted in italics in Q_{19} .

Fourth, to fulfill Requirement 2, we need to add a primary diagnosis column to the raw data that are retrieved by query Q_{19} and directly produce the feature value being examined. To fulfill Requirement 3, we need to sort the retrieved raw data in reverse chronological order. To meet both demands, we transform Q_{19} to the following final lineage tracing query:

```
 $Q_{20}$ : select cast(e.admit_time as date) as visit_date,
        case when m.diagnosis_code is null then null
        else m.dx_code_description || '(' || m.diagnosis_code || ')'
        end as primary_diagnosis,
        e.department, e.admitting_provider, e.facility
from encounter e
    inner join asthma_encounter_id e_id
        on e.encounter_id = e_id.encounter_id
    left outer join diagnosis d
        on e.encounter_id = d.encounter_id
    left outer join diagnosis_code_master m
        on d.diagnosis_code = m.diagnosis_code and d.ICD_version = m.ICD_version
where e.admit_time between today_date - 365 and today_date
    and e.encounter_type = 'emergency'
    and e.patient_id = asthma_patient_id
    and d.dx_sequence_number = 1
        -- primary diagnosis
order by e.admit_time desc;
```

The differences between Q_{19} and Q_{20} are highlighted in italics in Q_{20} .

Example 4: The total number of distinct medications ordered for the patient in the prior 12 months

As defined by query Q_4 in the “Intermediate result tables” section, the intermediate result table *med_features_1* contains 2 temporal features. One of them is the total number of distinct medications ordered for the patient in the prior 12 months. To form the lineage tracing query for a value of this feature, we proceed as follows.

First, we remove the other feature from query Q_4 and obtain the following simplified query:

```
 $Q_{21}$ : select patient_id,
        count(distinct medication_id) as count_distinct_medications_ordered
from ordered_medication
where ordering_time between today_date - 365 and today_date
group by patient_id;
```

Second, to address Reason 2 given in the “Requirement 1” section, we should include from the *ordered_medication* table only its attributes essential for automatic explanation. Using a modified version of Cui *et al.*'s lineage tracing techniques [23,37] that meets this demand, we form a preliminary lineage tracing query Q_{22} based on query Q_{21} and *asthma_patient_id*:

Q₂₂: *select medication_id, ordering_time, quantity, dose_unit, refills, ordering_provider, end_time*
from ordered_medication
where ordering_time between today_date - 365 and today_date
and patient_id = asthma_patient_id;

The differences between *Q₂₁* and *Q₂₂* are highlighted in italics in *Q₂₂*.

Third, to fulfill Sub-requirement 4.2, we could retrieve only the most recent order of each distinct medication ordered for the patient in the prior 12 months as the lineage information. This is done by transforming query *Q₂₂* to the following query:

Q₂₃: *select medication_id, ordering_time, quantity, dose_unit, refills, ordering_provider, end_time*
from (select medication_id, ordering_time, quantity, dose_unit, refills, ordering_provider, end_time,
row_number() over(partition by medication_id order by ordering_time desc) as row_sequence_number
from ordered_medication
where ordering_time between today_date - 365 and today_date
and patient_id = asthma_patient_id
) b
where row_sequence_number = 1;

The differences between *Q₂₂* and *Q₂₃* are highlighted in italics in *Q₂₃*.

Fourth, to fulfill Requirement 2, we add a medication name column to the raw data that are retrieved by query *Q₂₃* and directly produce the feature value being examined. To fulfill Requirement 3, we sort the retrieved raw data in reverse chronological order. To meet both demands, we transform *Q₂₃* to the following final lineage tracing query:

Q₂₄: *select o.ordering_time, m.name as medication_name, o.quantity, o.dose_unit, o.refills, o.ordering_provider, o.end_time*
from (select medication_id, ordering_time, quantity, dose_unit, refills, ordering_provider, end_time
from (select medication_id, ordering_time, quantity, dose_unit, refills, ordering_provider, end_time,
row_number() over(partition by medication_id order by ordering_time desc) as
row_sequence_number
from ordered_medication
where ordering_time between today_date - 365 and today_date
and patient_id = asthma_patient_id
) b
where row_sequence_number = 1
) o,
medication_master m
where o.medication_id = m.medication_id
order by o.ordering_time desc;

The differences between *Q₂₃* and *Q₂₄* are highlighted in italics in *Q₂₄*.

Some considerations for future computer coding implementation of our proposed automated lineage tracing approach

Maximizing the automation degree of the lineage tracing query formation process

For a select-project-join-aggregate materialized view, Cui *et al.* [23,37] used a fully automated approach to analyze its definition query to derive a lineage tracing query for a tuple in it. In our case of automatically explaining machine learning predictions, all temporal features used for making predictions and automatic explanation are known at machine learning model building time. In general, for each temporal feature, we can form a lineage tracing query either manually or semi-automatically, but often not fully automatically, beforehand. Nevertheless, once the query is formed and put into the knowledge base of the automated explaining function, we can use the query to automatically retrieve the lineage information of a value of the feature at prediction time.

As mentioned before, automatic explanation poses several unique requirements on automated lineage tracing. Two of them make it difficult to fully automate the lineage tracing query formation process. First, Requirement 1 says that the lineage information retrieved for a temporal feature value should include only a small set of relevant attributes specific to the temporal feature. Almost infinite attributes and temporal features could possibly be used for clinical machine learning. Thus, it is infeasible to pre-compile the set of relevant attributes for every possible temporal feature. Second, Requirement 2 says that when acquiring the lineage of a value for certain temporal features, we need to include some attributes that are specific to the

temporal feature and do not directly produce the feature value. For a reason similar to the above, it is infeasible to pre-compile the set of such attributes for every possible such temporal feature.

Although we cannot fully automate the lineage tracing query formation process in the most general case, we can still use 2 methods to maximize the process' automation degree and reduce the workload of the developers of the automated explaining function. First, for a temporal feature, we can use an approach similar to that in Cui *et al.* [23,37] to automatically form a draft lineage tracing query. The developers of the automated explaining function revise this query as needed to obtain the final lineage tracing query. Second, the same temporal feature is often used for multiple predictive modeling tasks. We can create a library of lineage tracing queries for temporal features to facilitate query reuse across various predictive modeling tasks. This library is formed for a data set in the Observational Medical Outcomes Partnership (OMOP) common data model format [50] using its linked standardized terminologies [51]. This format standardizes administrative and clinical variables from ≥ 10 large U.S. healthcare systems [52,53]. For any data set that is put into this format, we can use this library to obtain lineage tracing queries.

Improving the lineage tracing speed

As mentioned before, the user of the automated explaining function wants the lineage tracing process for a temporal feature value to be finished quickly, preferably within 1 second. To expedite tracing the lineage of a tuple in a materialized view defined by a select-project-join-aggregate query S , Cui *et al.* [23,37,49] advocated creating a materialized view for each intermediate select-project-join-aggregate segment of the canonical form of the logical query plan for S . While this boosts the lineage tracing speed, the resulting speed is still not fast enough to reach a sub-second response time [23,39]. To further improve the lineage tracing speed, we can build indices [39,42] on the selection and join attributes of both the base tables and the materialized views created for the intermediate select-project-join-aggregate segments. For instance, in Example 3, we can build 1 index on the *encounter_id* column of the materialized view *asthma_encounter_id*, and another index on the *patient_id* column of the *encounter* base table. We can create indices either manually or by using an automated index design tool provided by a commercial relational database system [54-56]. Typically, each intermediate result table containing 1 or more temporal features is computed on 1 or a few base tables using no more than a small number of join operations. The lineage tracing query for a temporal feature value falls into a similar case. Thus, with appropriate indices, we would expect the lineage tracing query to finish execution quickly. For base tables of moderate sizes and simple materialized views, Cui and Widom [39] showed that lineage tracing can be done within 1 second when indices exist on the keys of the base tables. For large base tables and temporal features computed through more complex procedures, we would expect that more indices are needed to reach a sub-second response time.

The above discussion focuses on the case that the electronic medical record data are stored in a relational database and features are extracted using SQL queries. When the electronic medical record data are stored in a big data system and features are extracted using map and reduce functions [44] or Pig Latin [46], we can modify the corresponding existing lineage tracing techniques [42,43,45] in a similar way to enable lineage tracing to aid automatically explaining machine learning predictions for clinical decision support.

Directions for future research

The above discussion describes the high-level design approach for our proposed automated lineage tracing module. To complete the detailed design of the proposed automated lineage tracing approach, implement the module in computer code, and test the module's performance, much research is needed along the following directions:

- 1) We need to compile a list of attributes and temporal feature types most commonly used in building clinical machine learning predictive models. For these attributes and temporal feature types, we need to complete the detailed design and the computer coding implementation of the proposed automated lineage tracing approach.
- 2) We need to come up with an automated approach to design indices needed for improving the lineage tracing speed. The database research community has developed several automated index design approaches [54-56]. We can modify these approaches to fit the database querying workload posed by automated lineage tracing.
- 3) We plan to assess the execution speed of the proposed automated lineage tracing approach after implementing it in computer code.
- 4) As shown by the lots of prior work on automated lineage tracing shown in the "An overview of the existing automated lineage tracing techniques" section, the database research community takes it for granted that automated lineage tracing could help users better understand the data and save time in doing data analysis. To the best of our knowledge, no formal study has been published on measuring the impact of automated lineage tracing on users' data analysis and decision making process. After implementing our proposed automated lineage tracing module, we plan to choose several clinical predictive modeling tasks and assess for each task, the impact of offering the module on the data analysis and decision making process of the users of the automated explaining function. In particular, we plan to evaluate whether the addition of the module

benefits the user and improves outcomes, e.g., by saving the user's time, making it easier for the user to understand the predictions given by the machine learning predictive model, and helping the user better understand the patient's situation and make better clinical decisions.

Limitations of our proposed automated lineage tracing approach

Our proposed automated lineage tracing approach has several limitations:

- 1) To build clinical machine learning predictive models, we usually use temporal features that are computed by SQL queries of low or moderate complexities. It is possible that some temporal features used to build certain predictive models are computed by rather complex SQL queries. We may not be able to finish the lineage tracing process for a value of such a temporal feature quickly, regardless of how many indices are built to expedite this process. For example, this could happen if the SQL query uses complex procedural code, which has no property that can be used to simplify the lineage tracing process [39]. Having a long lineage tracing time could make the user of the automated explaining function become impatient. Nevertheless, it is still faster and more convenient to do lineage tracing using our automated approach than to let the user do manual drill-through.
- 2) Our proposed automated lineage tracing approach works for any feature values computed by the standard aggregation functions in SQL on longitudinal structured data. For certain deep learning predictive models built on longitudinal structured data, we could use our previously proposed method [16] to semi-automatically extract comprehensible and predictive temporal features from the models and the longitudinal structured data, and then apply our automated approach to trace the lineage of the values of these features. For any other deep learning predictive model that is built directly on longitudinal structured data and uses incomprehensible features hidden in the neurons of the deep neural network, we can no longer use our automated approach to trace the lineage of the values of these features.
- 3) Almost infinite attributes and temporal features could possibly be used for clinical machine learning. Also, some attributes are not covered by the OMOP common data model. For the reasons given in the "Maximizing the automation degree of the lineage tracing query formation process" section, we could maximize the automation degree of the lineage tracing query formation process for only certain types of temporal features formed on certain attributes. For any other temporal feature, the developers of the automated explaining function could still need a non-trivial amount of time to create the corresponding lineage tracing query.

Conclusions

Automatically explaining machine learning predictions is critical to overcome the model interpretability barrier to using machine learning predictive models in clinical practice. Our previously developed automatic explanation method for machine learning predictions can be used to address this barrier, but a gap remains to fulfill the need of rapidly drilling through a feature value in an explanation that is computed by an aggregation function on the raw data. This paper articulates this gap and outlines an automated lineage tracing approach to close the gap, providing a roadmap for future research. By offering the automated drill-through capability, we intend to help the user of the automated explaining function save time, better understand the patient's situation, and make better clinical decisions. It would take several people multiple years to work out the detailed design and the computer coding implementation of the proposed automated lineage tracing approach. We hope this paper will make some researchers become interested in and join the research endeavor on this topic. Only after the detailed design and the computer coding implementation of the proposed automated lineage tracing approach are fully worked out, we could deploy the automated lineage tracing module in clinical practice and measure the module's impact on clinicians' decision making process. The principle of our automated lineage tracing approach generalizes to non-medical data and other automated methods to explain machine learning predictions.

Acknowledgments

We thank Xiaoyi Zhang and Brian Kelly for useful discussions. GL was partially supported by the National Heart, Lung, and Blood Institute of the National Institutes of Health under Award Number R01HL142503. The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

Conflicts of interest

None declared.

Abbreviations

ED: emergency department

HMC: Harborview Medical Center

ICD-10: International Classification of Diseases, Tenth Revision

OMOP: Observational Medical Outcomes Partnership
SQL: Structured Query Language
UWMC: University of Washington Medical Center

References

1. Kaggle homepage. <https://www.kaggle.com>, 2021.
2. Steyerberg EW. *Clinical Prediction Models: A Practical Approach to Development, Validation, and Updating*, 2nd ed. New York, NY: Springer; 2019. ISBN:3030163989
3. Lee G, Wang S, Dipuro F, Hou J, Grover P, Low LL, Liu N, Loke CY. Leveraging on predictive analytics to manage clinic no show and improve accessibility of care. In: *Proceedings of 2017 IEEE International Conference on Data Science and Advanced Analytics*. 2017 Presented at: DSAA'17; October 19-21, 2017; Tokyo, Japan p. 429-438. doi:10.1109/DSAA.2017.25
4. Dean NC, Jones BE, Jones JP, Ferraro JP, Post HB, Aronson D, Vines CG, Allen TL, Haug PJ. Impact of an electronic clinical decision support tool for emergency department patients with pneumonia. *Ann Emerg Med* 2015;66(5):511-520. PMID:25725592
5. Hsu JC, Chen YF, Chung WS, Tan TH, Chen T, Chiang JY. Clinical verification of a clinical decision support system for ventilator weaning. *Biomed Eng Online* 2013;12 Suppl 1:S4. PMID:24565021
6. Barbieri C, Molina M, Ponce P, Tothova M, Cattinelli I, Ion Titapiccolo J, Mari F, Amato C, Leipold F, Wehmeyer W, Stuard S, Stopper A, Canaud B. An international observational study suggests that artificial intelligence for clinical decision support optimizes anemia management in hemodialysis patients. *Kidney Int* 2016;90(2):422-429. PMID:27262365
7. Brier ME, Gaweda AE, Dailey A, Aronoff GR, Jacobs AA. Randomized trial of model predictive control for improved anemia management. *Clin J Am Soc Nephrol* 2010;5(5):814-820. PMID:20185598
8. Gaweda AE, Aronoff GR, Jacobs AA, Rai SN, Brier ME. Individualized anemia management reduces hemoglobin variability in hemodialysis patients. *J Am Soc Nephrol* 2014;25(1):159-166. PMID: 24029429
9. Gaweda AE, Jacobs AA, Aronoff GR, Brier ME. Model predictive control of erythropoietin administration in the anemia of ESRD. *Am J Kidney Dis* 2008;51(1):71-79. PMID:18155535
10. Hamlet KS, Hobgood A, Hamar GB, Dobbs AC, Rula EY, Pope JE. Impact of predictive model-directed end-of-life counseling for Medicare beneficiaries. *Am J Manag Care* 2010;16(5):379-384. PMID:20469958
11. Luo G. Automatically explaining machine learning prediction results: a demonstration on type 2 diabetes risk prediction. *Health Inf Sci Syst* 2016;4:2. PMID:26958341
12. Luo G, Johnson MD, Nkoy FL, He S, Stone BL. Automatically explaining machine learning prediction results on asthma hospital visits in asthmatic patients: secondary analysis. *JMIR Med Inform* 2020;8(12):e21965. PMID:33382379
13. Tong Y, Messinger AI, Luo G. Testing the generalizability of an automated method for explaining machine learning predictions on asthma patients' asthma hospital visits to an academic healthcare system. *IEEE Access* 2020;8:195971-195979. PMID:33240737
14. Luo G, Nau CL, Crawford WW, Schatz M, Zeiger RS, Koebnick C. Generalizability of an automatic explanation method for machine learning prediction results on asthma-related hospital visits in patients with asthma: quantitative analysis. *J Med Internet Res* 2021;23(4):e24153. PMID:33856359
15. Halamka JD. Early experiences with big data at an academic medical center. *Health Aff (Millwood)*. 2014;33(7):1132-1138. PMID:25006138
16. Luo G. A roadmap for semi-automatically extracting predictive and clinically meaningful temporal features from medical data for predictive modeling. *Glob Transit* 2019;1:61-82. PMID:31032483
17. Luo G, Nau CL, Crawford WW, Schatz M, Zeiger RS, Rozema E, Koebnick C. Developing a predictive model for asthma-related hospital encounters in patients with asthma in a large, integrated health care system: secondary analysis. *JMIR Med Inform*, 2020;8(11):e22689. PMID:33164906
18. Tong Y, Messinger AI, Wilcox AB, Mooney SD, Davidson GH, Suri P, Luo G. Forecasting future asthma hospital encounters of patients with asthma in an academic health care system: predictive model development and secondary analysis study. *J Med Internet Res* 2021;23(4):e22796.
19. Luo G, He S, Stone BL, Nkoy FL, Johnson MD. Developing a model to predict hospital encounters for asthma in asthmatic patients: secondary analysis. *JMIR Med Inform* 2020;8(1):e16080. PMID:31961332
20. Garcia-Molina H, Ullman JD, Widom J. *Database Systems: the Complete Book*, 2nd ed. Upper Saddle River, NJ: Pearson; 2008. ISBN:0131873253
21. Cunningham C, Graefe G, Galindo-Legaria CA. PIVOT and UNPIVOT: optimization and execution strategies in an RDBMS. In: *Proceedings of the 30th International Conference on Very Large Data Bases*, 2004 Presented at: VLDB'04; August 31 - September 3, 2004; Toronto, Canada p. 998-1009.

22. Lyman JA, Scully K, Harrison JH Jr. The development of health care data warehouses to support data mining. *Clin Lab Med* 2008;28(1):55-71. PMID:18194718
23. Cui Y, Widom J. Practical lineage tracing in data warehouses. In: *Proceedings of the 16th International Conference on Data Engineering*. 2000 Presented at: ICDE'00; February 28-March 3, 2000; San Diego, CA p. 367-378. doi:10.1109/ICDE.2000.839437
24. Liu B, Hsu W, Ma Y. Integrating classification and association rule mining. In: *Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining*. 1998 Presented at: KDD'98; August 27-31, 1998; New York City, NY p. 80-86.
25. Fayyad UM, Irani KB. Multi-interval discretization of continuous-valued attributes for classification learning. In: *Proceedings of the 13th International Joint Conference on Artificial Intelligence*. 1993 Presented at: IJCAI'93; August 28-September 3, 1993; Chambéry, France p. 1022-1029.
26. Thabtah FA. A review of associative classification mining. *Knowledge Eng Review* 2007;22(1):37-65. doi:10.1017/S0269888907001026
27. Alaa AM, van der Schaar M. Prognostication and risk factors for cystic fibrosis via automated machine learning. *Sci Rep* 2018;8(1):11242. PMID:30050169
28. Alaa AM, van der Schaar M. AutoPrognosis: automated clinical prognostic modeling via Bayesian optimization with structured kernel learning. In: *Proceedings of 35th International Conference on Machine Learning*. 2018 Presented at: ICML'18; July 10-15, 2018; Stockholm, Sweden p. 139-148.
29. Molnar C. *Interpretable Machine Learning*. Morrisville, NC: lulu.com; 2020. ISBN:0244768528
30. Guidotti R, Monreale A, Ruggieri S, Turini F, Giannotti F, Pedreschi D. A survey of methods for explaining black box models. *ACM Comput Surv* 2019;51(5):93. doi:10.1145/3236009
31. Rudin C, Shaposhnik Y. Globally-consistent rule-based summary-explanations for machine learning models: application to credit-risk evaluation. In: *Proceedings of INFORMS 11th Conference on Information Systems and Technology*. 2019 Presented at: CIST'19; October 19-20, 2019; Seattle, WA p. 1-19. doi:10.2139/ssrn.3395422
32. Ribeiro MT, Singh S, Guestrin C. Anchors: high-precision model-agnostic explanations. In: *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*. 2018 Presented at: AAAI'18; February 2-7, 2018; New Orleans, LA p. 1527-1535.
33. Ikeda R, Widom J. Data lineage: a survey. Stanford University technical report. http://ilpubs.stanford.edu:8090/918/1/lin_final.pdf, 2009.
34. Cheney J, Chiticariu L, Tan WC. Provenance in databases: why, how, and where. *Found Trends Databases* 2009;1(4):379-474. doi:10.1561/19000000006
35. Simmhan Y, Plale B, Gannon D. A survey of data provenance in e-science. *SIGMOD Rec* 2005;34(3):31-36. doi:10.1145/1084805.1084812
36. Bose R, Frew J. Lineage retrieval for scientific data processing: a survey. *ACM Comput Surv* 2005;37(1):1-28. doi:10.1145/1057977.1057978
37. Cui Y, Widom J, Wiener JL. Tracing the lineage of view data in a warehousing environment. *ACM Trans Database Syst* 2000;25(2):179-227. doi:10.1145/357775.357777
38. Gupta A, Mumick IS. *Materialized Views: Techniques, Implementations, and Applications*. Cambridge, MA: The MIT Press; 1999. ISBN:0262571226
39. Cui Y, Widom J. Lineage tracing for general data warehouse transformations. *VLDB J* 2003;12(1):41-58. doi:10.1007/s00778-002-0083-8
40. Ikeda R, Sarma AD, Widom J. Logical provenance in data-oriented workflows. In: *Proceedings of the 29th IEEE International Conference on Data Engineering*. 2013 Presented at: ICDE'13; April 8-12, 2013; Brisbane, Australia p. 877-888. doi:10.1109/ICDE.2013.6544882
41. Zhang M, Zhang X, Zhang X, Prabhakar S. Tracing lineage beyond relational operators. In: *Proceedings of the 33rd International Conference on Very Large Data Bases*. 2007 Presented at: VLDB'07; September 23-27, 2007; Vienna, Austria p. 1116-1127.
42. Ikeda R, Park H, Widom J. Provenance for generalized map and reduce workflows. In: *Proceedings of the 5th Biennial Conference on Innovative Data Systems Research*. 2011 Presented at: CIDR'11; January 9-12, 2011; Asilomar, CA p. 273-283.
43. Park H, Ikeda R, Widom J. RAMP: a system for capturing and tracing provenance in MapReduce workflows. *Proc VLDB Endow* 2011;4(12):1351-1354. doi:10.14778/3402755.3402768
44. Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters. In: *Proceedings of the 6th Symposium on Operating System Design and Implementation*. 2004 Presented at: OSDI'04; December 6-8, 2004; San Francisco, CA p. 137-150.
45. Amsterdamer Y, Davidson SB, Deutch D, Milo T, Stoyanovich J, Tannen V. Putting Lipstick on Pig: enabling database-style workflow provenance. *Proc VLDB Endow* 2011;5(4):346-357. doi:10.14778/2095686.2095693

46. Olston C, Reed B, Srivastava U, Kumar R, Tomkins A. Pig Latin: a not-so-foreign language for data processing. In: Proceedings of the ACM SIGMOD International Conference on Management of Data. 2008 Presented at: SIGMOD'08; June 10-12, 2008; Vancouver, BC, Canada p. 1099-1110. doi:10.1145/1376616.1376726
47. Buneman P, Chapman A, Cheney J. Provenance management in curated databases. In: Proceedings of the ACM SIGMOD International Conference on Management of Data. 2006 Presented at: SIGMOD'06; June 27-29, 2006; Chicago, IL p. 539-550. doi:10.1145/1142473.1142534
48. Schelter S, Böse J, Kirschnick J, Klein T, Seufert S. Automatically tracking metadata and provenance of machine learning experiments. In: Proceedings of the ML Systems Workshop at NIPS 2017. 2017 Presented at: ML Systems Workshop at NIPS'17; December 8, 2017; Long Beach, CA p. 1-8.
49. Cui Y, Widom J. Storing auxiliary data for efficient maintenance and lineage tracing of complex views. In: Proceedings of the Second Intl Workshop on Design and Management of Data Warehouses. 2000 Presented at: DMDW'00; June 5-6, 2000; Stockholm, Sweden Article 11.
50. Observational Health Data Sciences and Informatics data standardization homepage. <https://www.ohdsi.org/data-standardization>, 2021.
51. Observational Health Data Sciences and Informatics standardized vocabularies homepage. <https://www.ohdsi.org/web/wiki/doku.php?id=documentation:vocabulary:sidebar>, 2021.
52. Hripcsak G, Duke JD, Shah NH, Reich CG, Huser V, Schuemie MJ, Suchard MA, Park RW, Wong ICK, Rijnbeek PR, van der Lei J, Pratt N, Norén GN, Li Y, Stang PE, Madigan D, Ryan PB. Observational Health Data Sciences and Informatics (OHDSI): opportunities for observational researchers. *Stud Health Technol Inform* 2015;216:574-578. PMID:26262116
53. Overhage JM, Ryan PB, Reich CG, Hartzema AG, Stang PE. Validation of a common data model for active safety surveillance research. *J Am Med Inform Assoc* 2012;19(1):54-60. PMID:22037893
54. Das S, Grbic M, Ilic I, Jovandic I, Jovanovic A, Narasayya VR, Radulovic M, Stikic M, Xu G, Chaudhuri S. Automatically indexing millions of databases in Microsoft Azure SQL database. In: Proceedings of the ACM SIGMOD International Conference on Management of Data. 2019 Presented at: SIGMOD'19; June 30-July 5, 2019; Amsterdam, Netherlands p. 666-679. doi:10.1145/3299869.3314035
55. Dageville B, Das D, Dias K, Yagoub K, Zaït M, Ziauddin M. Automatic SQL tuning in Oracle 10g. In: Proceedings of the 30th International Conference on Very Large Data Bases. 2004 Presented at: VLDB'04; August 31-September 3, 2004; Toronto, Canada p. 1098-1109. doi:10.1016/B978-012088469-8.50096-6
56. Zilio DC, Rao J, Lightstone S, Lohman GM, Storm AJ, Garcia-Arellano C, Fadden S. DB2 Design Advisor: integrated automatic physical database design. In: Proceedings of the 30th International Conference on Very Large Data Bases. 2004 Presented at: VLDB'04; August 31-September 3, 2004; Toronto, Canada p. 1087-1097. doi:10.1016/B978-012088469-8.50095-4