Learning-Based Relaxation of Completeness Requirements for Data Entry Forms

HICHEM BELGACEM, University of Luxembourg, Luxembourg
XIAOCHEN LI*, Dalian University of Technology, China
DOMENICO BIANCULLI, University of Luxembourg, Luxembourg
LIONEL BRIAND, University of Luxembourg, Luxembourg and University of Ottawa, Canada

Data entry forms use completeness requirements to specify the fields that are required or optional to fill for collecting necessary information from different types of users. However, because of the evolving nature of software, some required fields may not be applicable for certain types of users anymore. Nevertheless, they may still be incorrectly marked as required in the form; we call such fields obsolete required fields. Since obsolete required fields usually have "not-null" validation checks before submitting the form, users have to enter meaningless values in such fields in order to complete the form submission. These meaningless values threaten the quality of the filled data, and could negatively affect stakeholders or learning-based tools that use the data. To avoid users filling meaningless values, existing techniques usually rely on manually written rules to identify the obsolete required fields and relax their completeness requirements. However, these techniques are ineffective and costly.

In this paper, we propose LACQUER, a learning-based automated approach for relaxing the completeness requirements of data entry forms. LACQUER builds Bayesian Network models to automatically learn conditions under which users had to fill meaningless values. To improve its learning ability, LACQUER identifies the cases where a required field is only applicable for a small group of users, and uses SMOTE, an oversampling technique, to generate more instances on such fields for effectively mining dependencies on them. During the data entry session, LACQUER predicts the completeness requirement of a target based on the already filled fields and their conditional dependencies in the trained model.

Our experimental results show that LACQUER can accurately relax the completeness requirements of required fields in data entry forms with precision values ranging between 0.76 and 0.90 on different datasets. LACQUER can prevent users from filling 20% to 64% of meaningless values, with negative predictive values (i.e., the ability to correctly predict a field as "optional") between 0.72 and 0.91. Furthermore, LACQUER is efficient; it takes at most 839 ms to predict the completeness requirement of an instance.

CCS Concepts: • **Computing methodologies** → *Bayesian network models*; • **Information systems** → **Recommender systems**; • **Software and its engineering** → **Software usability**.

Additional Key Words and Phrases: Form filling, Data entry forms, Completeness Requirements Relaxation, Machine Learning, Software data quality, User interfaces

---

*Part of this work was done while the author was affiliated with the University of Luxembourg, Luxembourg.

Authors' addresses: Hichem Belgacem, University of Luxembourg, Luxembourg, hichem.belgacem@uni.lu; Xiaochen Li, Dalian University of Technology, China, xiaochen.li@dlut.edu.cn; Domenico Bianculli, University of Luxembourg, Luxembourg, domenico.bianculli@uni.lu; Lionel Briand, University of Luxembourg, Luxembourg and University of Ottawa, Canada, lionel.briand@uni.lu.

---

1 INTRODUCTION

Software designers use data entry forms to collect inputs of users who interact with software systems [27, 45]. To correctly collect the necessary information from users, designers typically define the completeness requirements of fields in data entry forms. These completeness requirements specify the fields that are required or optional to fill for different types of users.

However, as the software system and the application requirements change, data entry forms change too. Such changes may result in some fields, previously marked as required, becoming inapplicable for certain types of users. We call *obsolete required fields* the fields whose "required" attribute does not remain valid with respect to the current application requirements. Although such fields are set as "required" in the form, the correct completeness requirement should be "optional".

When obsolete required fields are included in an input form, since the system usually has client-side validation checks [51] to ensure that all the required fields have been filled in, users are obliged to fill the required fields with meaningless values (e.g., "@", "n/a") to be able to submit the form [3, 30]. We have observed this phenomenon both on a popular biomedical information collection platform NCBI [4], in which more than half of the users have filled meaningless values in required fields, and in a dataset provided by our industrial partner in the financial domain.

Obsolete required fields represent an extra burden for the users, costing additional time for filling the input form, and might lead to users interrupting the data entry process, with potential loss of business opportunities (e.g., a prospective client giving up during the registration phase due to the complexity of the input form). Moreover, the meaningless values filled through these obsolete required fields are then transferred to the software system using them and may affect the overall data quality of the system [37]. For example, given a categorical field (which is an obsolete required field), the user can choose the first value in a combo box just to skip filling this field. Even though the value is chosen from the list of candidate values, this value is meaningless since the field should not be filled at the beginning [3]. This value can be used as an input by machine learning-based tools (for example an automated form filling tool [5]), which can then lead to more errors (e.g., wrongly predicting the values of some fields).

To automatically relax completeness requirements and avoid meaningless values, existing work has proposed adaptive form tools [6, 19, 49], which enable form designers to set required fields as optional when certain conditions hold. These tools first require form designers to define a *complete and final* set of completeness requirements, capturing the conditions for which a field should be required or optional. Then, they use intermediate representations such as XML [6] and dynamic condition response graphs [49] to represent the completeness requirements rules and implement adaptive behaviors. In addition, there are commercial tools (e.g., Gravity Forms [42], Google Forms [25]) that assist designers in designing adaptive forms, where fields can be displayed or hidden based on the value of already filled fields in the form. Similar to existing research approaches, these commercial tools assume that designers already have a *complete and final* set of completeness requirements describing the adaptive behaviour of the form during the design phase.

However, due to the complexity of the domain (with hundreds of fields) and the evolution of the software, identifying a priori a comprehensive set of completeness requirements is not a viable solution. Moreover, even if they could be identified, such completeness requirements could become quickly obsolete, limiting the use of existing adaptive form tools.

To solve this problem, we propose LACQUER, a <u>L</u>earning-b<u>A</u>sed <u>C</u>ompleteness re<u>QU</u>ir<u>E</u>ments <u>R</u>elaxation approach, to automatically learn the conditions under which completeness requirements can be relaxed (i.e., when a required field can become optional). The basic idea of LACQUER is to build machine learning models to learn the conditions under which users had to fill meaningless values based on the data provided as input in past data entry sessions (hereafter called *historical*

input instances). Using these models, the already-filled fields in a data entry form can then be used as features to predict whether a required field should become optional for certain users. LACQUER can be used during the form filling process to refactor data entry forms by dynamically removing obsolete required fields at run time, helping designers identify completeness requirements that should be relaxed.

LACQUER includes three phases: *model building*, *form filling relaxation*, and *threshold determination*. Given a set of historical input instances, the model building phase identifies the meaningless values filled by users and builds Bayesian network (BN) models to represent the completeness requirement dependencies among form fields (i.e., the conditions upon which users fill meaningless values). To improve its learning ability, LACQUER identifies also the cases where a required field is only applicable for a small group of users; it uses the synthetic minority oversampling technique SMOTE to generate more instances on such fields for effectively mining dependencies on them. Once the trained models are available, during the data entry session, the form filling relaxation phase predicts the completeness requirement of a target field based on the values of the already-filled fields and their conditional dependencies in the trained models. The predicted completeness requirement of a field and the corresponding predicted probability (endorsed based on a "threshold" automatically determined) are then used to implement adaptive behaviors of data entry forms.

The overall architecture of LACQUER has been inspired by LAFF [5], our previous work on automated form filling of data entry forms. The main similarities between these approaches derive from their shared challenges associated with the application domain (form filling). These challenges include (1) the arbitrary filling order and (2) partially filled forms. To address the first challenge, similar to LAFF, we use BNs in order to mine the relationships between filled fields and the target field to avoid training a separate model for each filling order. As for the second challenge, once again similar to LAFF, we use an endorser module to avoid providing inaccurate suggestions to the user when the form does not contain enough information for the model. More details about the similarities and differences between LACQUER and LAFF are provided in section 6.

We evaluated LACQUER using form filling records from both a public dataset and a proprietary dataset extracted from a production-grade enterprise information system in the financial domain. The experimental results show that LACQUER can accurately relax the completeness requirements of required fields in data entry forms with a precision value between 0.76 and 0.90 when predicting the truly required fields. In a sequential filling scenario, i.e., when users fill data entry forms in the default order determined by the form tab sequence, LACQUER can prevent users from providing meaningless values in 20% to 64% of the cases, with a negative predictive value (representing the ability of LACQUER to correctly predict a field as "optional") between 0.72 and 0.91, significantly outperforming state-of-the-art rule-based approaches by 12 pp to 70 pp (with pp = percentage points) on the two datasets. Furthermore, LACQUER is efficient; it takes at most 839 ms to determine the completeness requirement of an input instance of the proprietary dataset.

To summarize, the main contributions of this paper are:

- The LACQUER approach, which addresses the problem of automated completeness requirements relaxation — an important challenge in designing data entry forms. To the best of our knowledge, LACQUER is the first work to combine BNs with oversampling and a probability-based endorser to provide accurate completeness requirement suggestions.
- An extensive evaluation assessing the effectiveness and efficiency of LACQUER and comparing it with state-of-the-art baselines[1].

---

[1]The implementation of LACQUER and the scripts used for evaluation are available at https://figshare.com/s/0fdeae041e 728e6d0a01; see also section 5.6.

The rest of the paper is organized as follows. Section 2 provides a motivating example and explains the basic definitions of automated completeness requirements relaxation and its challenges. Section 3 introduces the basic machine learning algorithms used in this paper. Section 4 describes the different steps and the core algorithms of LACQUER. Section 5 reports on the evaluation of LACQUER. Section 6 surveys related work. Section 7 discusses the usefulness and practical implication of LACQUER. Section 8 concludes the paper.

2 COMPLETENESS REQUIREMENT RELAXATION FOR DATA ENTRY FORMS

In this section, we introduce the concepts related to data entry forms, provide a motivating example, precisely define the problem of automated completeness requirement relaxation for data entry forms, and discuss its challenges.

2.1 Data Entry Forms

Data entry forms are composed of fields of different types, such as textual, numerical, and categorical. Textual and numerical fields collect free text and numerical values, respectively (e.g., the name and the age of a private customer of an energy provider); categorical fields provide a list of options from which users have to choose (e.g., nationality). Form developers can mark form fields either as *required* or *optional*, depending on the importance of the information to be collected. This decision is made during the design phase of the form based on the application completeness requirements. Such requirements capture the input data that shall be collected for certain types of users; they are fulfilled by setting the required/optional property of the corresponding fields in a data entry form. In other words, the required fields (also called mandatory fields [46]) of a form collect input information considered as important to the stakeholders who plan to use the collected information; the absence of this information could affect the application usage. On the contrary, optional fields collect information that is nice to have but whose absence is acceptable. For example, an energy provider cannot open a customer account when the customer name is missing; hence, the corresponding input field in a data entry form should be marked as "required". At the same time, an energy provider does not need to know the education level of a new private customer (though it could be useful for profiling), so the corresponding input field can be marked as "optional".

Some required fields can be further classified *conditionally required*, i.e., they are required only if certain conditions hold. For example, the field "marriage date" is required only if the value of the categorical field "civil status" is set to "married". Data entry forms that support "conditionally required fields" are generally called *adaptive forms* [6] or context-sensitive forms [3], since they exhibit adaptive behaviors based on the values filled by users. More specifically, these types of forms are programmed so that a field can be set from "required" to "optional" during the form-filling session, based on the input data; a change of this property also toggles the visibility of the field itself in the form. Such adaptive behaviours make the data entry form easier to use [3], since users can focus on the right fields they need to fill in.

Before submitting a data entry form, the form usually conducts a *client-side validation* check [51] — using some scripting language or built-in features of the environment where the form is visualized, like HTML attributes — to ensure that all the required fields have been filled in.

In this work, we consider a simple representation of an input form, with basic input fields that can have only a unique value that can be selected or entered, such as a text box (e.g., `<input type="text">` or `<textarea>` in HTML), a drop-down menu (e.g, `<select>` with single selection), or a radio button (e.g., `<input type="radio"` in HTML). This allows us to assume that a field can only have one completeness requirement; in other words, a field cannot be optional and required at the same time.

We do not support forms with more sophisticated controls or fields that can handle multiple selections (e.g., a checkbox group for multiple-choice answers or a drop-down menu with multiple selection), as often found in surveys and questionnaires. Note that in this case a field can be both optional and required at the same time, depending on the number of selected values in the group[2]. We plan to support this kind of complex controls as part of future work.

2.2 Motivating Example

Data entry forms are difficult to design [17] and subject to frequent changes [54]. These two aspects of data entry form design and development negatively impact the way developers deal with application completeness requirements in data entry forms.

For example, let us consider a data entry form in an energy provider information system, used for opening an account for business customers. For simplicity, we assume the form has only three required fields: "Company type" (categorical), "Field of activity" (categorical), and "Tax ID" (textual). Sometime after the deployment of the initial version of the system, the energy provider decides to support also the opening of customer accounts for non-profit organizations (NPOs). The developers update the form by adding (a) a new option "NPO" to the field "Company type", and (b) additional fields denoting information required for NPOs. After the deployment of the new form, a data entry operator of the energy provider (i.e., the end-user interfacing with the data entry form) notices a blocking situation when filling in the form for an NPO. Specifically, the form flags the field "Tax ID" as required; however, the company representative cannot provide one since the company is exempted from paying taxes. The clerk is then obliged to fill in the required field with a meaningless value (e.g., "@") to pass the validation check and be able to submit the form. Several weeks later, after noticing some issues in the generation of customers' reports, the data quality division of the energy provider reviews the data collected through the data entry form, detecting the presence of meaningless values. A subsequent meeting with IT analysts and developers reveals that those values have been introduced because the data entry form design has not been updated to take into account the new business requirements (i.e., opening accounts for NPOs) and the corresponding completeness requirements (i.e., some NPOs in certain fields of activity do not have a tax ID). For example, the current (but obsolete) form design always flags "tax ID" as a *required* field; however, when the "Company type" field is set to "NPO" and the "Field of activity" field is either "charity" or "education", the field "tax ID" should be *optional*.

These meaningless values filled during form filling negatively affect data quality [3], since they are considered as data entry errors and may lead to error propagation:

- **Data entry errors**: Users fill obsolete required fields with incorrect data (meaningless values) in order to proceed quickly in the workflow of the data-entry form [3].
- **Error propagation**: Meaningless value errors can propagate and create more errors [37], especially when these values are used in ML-based tools.

Meaningless value errors are difficult to identify because such values can pass all validation checks of the data entry form. A business may establish the practice of using specific values (e.g., "@" and "-1") when users do not need to fill some fields, as in the aforementioned example. However, even in this case the data quality team needs to carefully check the filled fields to ensure that all the data entry operators follow this convention, which is a time-consuming process.

Currently, there are some simple but rather impractical solutions to address the issue of filling meaningless values, including rule-based solution and dictionary-based solution:

---

[2]An example of complex input control is the case where users need to select at least three options from a multiple choice answer field (e.g., a checkbox group). Any option chosen before reaching the minimum number of selected values would be considered "required"; however, the same option chosen after the first three would be considered "optional".

- **Rule-based solution**: This solution defines for each field some rules capturing the conditions for which a required field can become optional, based on the values of the other form fields.
- **Dictionary-based solution**: This solution sets all fields containing meaningless values as optional. More specifically, the data quality division could first create a dictionary of meaningless values (e.g., "@", "$"). Users can then use such values when a field is not applicable in a certain form-filling scenario. Finally, the data quality division could analyze the historical input instances and mark a field as optional when users assign a value to it from the meaningless values dictionary. Such information could then be used to refactor the data entry form, setting the corresponding input field as optional.

However, the two solutions are not practical. Given the evolving nature of software [22, 52], the rule-based solution is not scalable and maintainable, especially when the number of fields (and their possible values, for categorical fields) increases. Moreover, as is the case for our industrial partner, it is difficult also for domain experts to formulate the completeness requirement of new fields, since they have to decide the exact impact of different field combinations on the new fields. Regarding the dictionary-based solution, the completeness requirement of a field usually depends on the values of other filled fields [3] (such as the aforementioned example of Tax ID), and cannot be detected only by looking at special/meaningless characters. This simple solution cannot help domain experts identify these useful conditions.

Therefore, we contend it is necessary to develop automated methods to learn such conditions directly from the data provided as input in past data entry sessions, so that completeness requirements of form fields can be automatically relaxed during new data entry sessions. Moreover, the learned conditions could also help designers identify completeness requirements that should be relaxed.

2.3 Problem Definition

In this paper, we deal with the problem of completeness requirement relaxation for data entry forms. The problem can be informally defined as deciding whether a required field in a form can be considered optional based on the values of the other fields and the values provided as input in previous data entry sessions for the same form. We formally define this problem as follows.

Let us assume we have a data entry form with n fields $F = \{f_1, f_2, \ldots, f_n\}$. Taking into account the required/optional attribute of each field, the set of fields can be partitioned into two groups: required fields (denoted by R) and optional fields (denoted by $\bar{R}$), where $\bar{R} \cup R = F$ and $\bar{R} \cap R = \emptyset$. Let VD represent a value domain that excludes empty values. Each field f_i in F can take a value from a domain V_i, where $V_i = VD_i$ if the field is required and $V_i = VD_i \cup \bot$ if the field is optional ($\bot$ is a special element representing an empty value).

Let $R^c \subseteq R$ be the set of conditionally required fields, which are required only when a certain condition $Cond$ is satisfied. For a field $f_k \in R^c$, we define the condition $Cond_k$ as the conjunction of predicates over the value of some other fields; more formally, $Cond_k = \bigwedge_{1 \leq i \leq n, i \neq k} h(f_i, v_i^c)$, where $f_i \in F, v_i^c \in V_i$, and h is a predicate over the field f_i with respect to the value v_i^c.

During form filling, at any time t the fields can be partitioned into two groups: fields that have been filled completely (denoted by C_t) and unfilled fields (denoted by $\bar{C}_t$); let G be the operation that extracts a field from a form during form filling $G(F) = f$, such that $(f \in C_t) \vee (f \in \bar{C}_t)$ and $C_t \cap \bar{C}_t = \emptyset$. By taking into account also the required/optional attribute, we have: filled required fields ($C_t \cap R$), filled optional fields ($C_t \cap \bar{R}$), unfilled required fields ($\bar{C}_t \cap R$), and unfilled optional fields ($\bar{C}_t \cap \bar{R}$).

When a form is about to be submitted (e.g., to be stored in a database), we define an *input instance* of the form to be $I^F = \{\langle f_1, v_1 \rangle, \ldots, \langle f_n, v_n \rangle\}$ with $f_i \in F$ and $v_i \in V_i$; we use the subscript t_j as
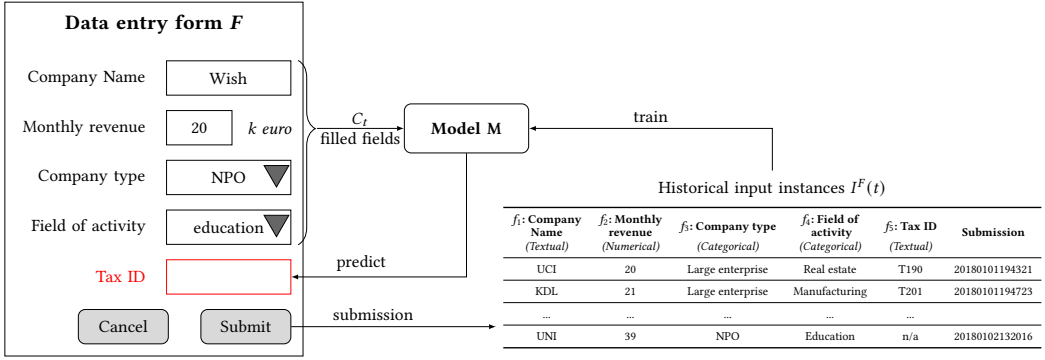
Fig. 1. The Automated Form Filling Relaxation Problem

in $I_{t_j}^F$ to denote that the input instance I^F was submitted at time t_j. We use the notation $I^F(t)$ to represent the set of *historical input instances* of the form that have been submitted up to a certain time instant t; $I^F(t) = \{I_{t_i}^F, I_{t_j}^F, \ldots, I_{t_k}^F\}$, where $t_i < t_j < t_k < t$. Hereafter, we drop the superscript F when it is clear from the context.

The completeness requirement relaxation problem can be defined as follows. Given a partially filled form $F = \{f_1, f_2, \ldots, f_n\}$ for which, at time t, we know $\bar{C}_t \neq \emptyset$, C_t, and R^c, a set of historical input instances $I^F(t)$, and a target field $f_p \in (R^c \cap \bar{C}_t)$ to fill, with $p \in 1 \ldots n$, we want to build a model M predicting whether, at time t, f_p should become optional based on C_t and $I^F(t)$.

Framing the problem definition scope. In this problem definition, our goal is *to relax the completeness requirements of a form* by determining which obsolete required fields should become optional to avoid filling meaningless values. We do not consider the case in which optional fields could become required; we leave extending LACQUER to automatically decide the completeness requirement of all fields as part of future work.

Moreover, as mentioned in the motivating example, in this definition, we mainly focus on the case of filling data entry forms from scratch. We do not consider the case in which an existing instance in the database is updated (including an update of the timestamp); for example, following our motivating example, if a company changes its "Field of activity" to "charity", then some fields like "tax ID" may become optional and do not need to be filled. LACQUER can be adapted to support this scenario and check if the completeness requirement of some fields need to be changed; we also leave this adaption as part of future work.

Application to the running example. Figure 1 is an example of a data entry form used to fill information needed to open an account for business customers with an energy provider The form F is composed of five fields, including f_1:"Company Name", f_2:"Monthly revenue", f_3:"Company type", f_4:"Field of activity", and f_5:"Tax ID". All the fields are initially required (i.e., $R = \{f_1, f_2, f_3, f_4, f_5\}$). Values filled in these fields are then stored in a database. An example of the database is shown on the right side of Figure 1. These values are collected during the data entry session with an automatically recorded timestamp indicating the submission time. Each row in the database represents an input instance (e.g., $I_{20180101194321}^F = \{\langle$"Company Name", UCI$\rangle, \ldots, \langle$"Tax ID", T190$\rangle\}$), where the column name corresponds to the field name in the form. The mapping can be obtained from the existing software design documentation or software implementation [5]. Using the data collected from different users, we can build a model M to learn possible relationships of completeness requirement

between different fields. Let us assume a scenario where during the creation of a customer account using F, the energy provider clerk has entered *Wish*, *20*, *NPO*, and *education* for fields f_1 to f_4, respectively. The field f_5 ("Tax ID") is the next field to be filled. Our goal is to automatically decide if field f_5 is required or not based on the values filled in fields f_1 to f_4.

2.4 Towards adaptive forms: challenges

Several tools for adaptive forms have been proposed [6, 19, 49]. These approaches use intermediate representations such as XML [6] and dynamic condition response graphs [49] to represent the completeness requirements rules and implement adaptive behaviours. Existing tools for adaptive forms usually assume that form designers already have, during the design phase, a *complete and final* set of completeness requirements, capturing the conditions for which a field should be required or optional.

However, this assumption is not valid in real-world applications. On one hand, data entry forms are not easy to design [17]. Data entry forms need to reflect the data that need to be filled in an application domain. Due to time pressure and the complexity of the domain (e.g., the number of fields needed to be filled and their interrelation), it is difficult to identify all the completeness requirements when designing the data entry form [2, 12]. On the other hand, data entry forms are subject to change: a recent study [54] has shown that 49% of web applications will modify their data constraints in a future version. The frequent changes in data constraints may also make the existing completeness requirements obsolete.

Hence the main challenge is how to create adaptive forms when the set of completeness requirements representing the adaptive behaviour of a form is incomplete and evolving.

3 PRELIMINARIES

Before illustrating our approach, we first briefly introduce two basic machine-learning algorithms we rely on.

3.1 Bayesian Networks

Bayesian networks (BNs) are probabilistic graphical models (PGM) in which a set of random variables and their conditional dependencies are encoded as a directed acyclic graph: nodes correspond to random variables and edges correspond to conditional probabilities.

The use of BNs for supervised learning [20] typically consists of two phases: structure learning and variable inference.

During *structure learning*, the graphical structure of the BN is automatically learned from a training set. First, the conditional probability between any two random variables is computed. Based on these probabilities, optimization-based search (e.g., hill climbing [21]) is applied to search the graphical structure. The search algorithm initializes a random structure, and then iteratively adds or deletes its nodes and edges to generate new structures. For each new structure, the search algorithm calculates a fitness function (e.g., Bayesian information criterion, BIC [40]) based on the nodes' conditional probabilities and on Bayes' theorem [20]. Structure learning stops when it finds a graphical structure that minimizes the fitness function.

Figure 2 shows an example of the BN structure learned based on the data submitted by the data entry form used in our example in section 2.2. This BN contains three nodes corresponding to three fields in the data entry form: variable *Revenue* depends on variable *Company type*; variable *Tax ID* depends on variables *Company type* and *Revenue*. For simplicity, we assume that the three variables are Boolean where a, b, and c denote that fields *Company type*, *Revenue*, and *Tax ID* are "required" respectively, and $\bar{a}$, $\bar{b}$, and $\bar{c}$ denote that these fields are optional.

			P(Revenue | Company type)			P(Tax ID | Company type, Revenue)			

The Bayesian Network structure shows *Company type* with arrows pointing to *Tax ID* and *Revenue*.

P(Company type)

a	ā
0.2	0.8

P(Revenue | Company type)

Company type	Revenue	
	b	b̄
a	0.4	0.6
ā	0.1	0.9

P(Tax ID | Company type, Revenue)

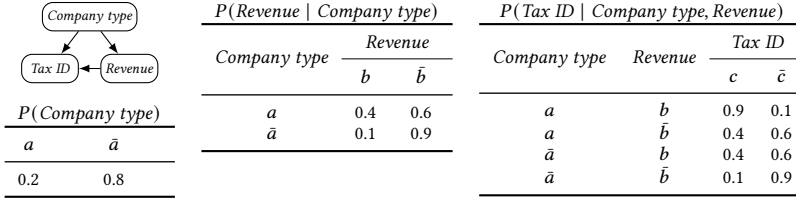| Company type	Revenue	Tax ID	
		c	c̄
a	b	0.9	0.1
a	b̄	0.4	0.6
ā	b	0.4	0.6
ā	b̄	0.1	0.9

Fig. 2. An Example of BN and the Probability Functions of its Nodes

In the PGM, each node is associated with a probability function (in this case, encoded as a table), which represents the conditional probability between the node and its parent(s). For example, in Figure 2 each variable has two values; the probability table for *Revenue* reflects the conditional probability $P(Revenue \mid Company\ type)$ between *Company type* and *Revenue* on these values.

Variable inference infers unobserved variables from the observed variables and the graphical structure of the BN using Bayes' theorem [20]. For example, we can infer the probability of *Tax ID* to be required (i.e., *Tax ID* = c) when the completeness requirement of *Company type* is required (denoted by $P(c \mid a)$) as follows:

$$P(c \mid a) = \frac{P(a,c)}{P(a)} = \frac{P(a,b,c) + P(a,\bar{b},c)}{P(a)}$$

$$= \frac{P(c \mid a,b)P(b \mid a)P(a) + P(c \mid a,\bar{b})P(\bar{b} \mid a)P(a)}{P(a)}$$

$$= \frac{0.9 * 0.4 * 0.2 + 0.4 * 0.6 * 0.2}{0.2} = 0.6$$

BNs have been initially proposed for learning dependencies among discrete random variables. They are also robust when dealing with missing observed variables; more specifically, variable inference can be conducted when some conditionally independent observed variables are missing [20]. Recently, they have been applied in the context of automated form filling [5].

3.2 Synthetic Minority Oversampling Technique (SMOTE)

A frequently encountered problem for training machine learning models using real-world data is that the number of instances per class can be imbalanced [32, 48]. To address this problem, many imbalanced learning approaches have been proposed in the literature. One of them is SMOTE [9]; it uses an oversampling method to modify the class distribution in a dataset (i.e., the ratio between instances in different classes). It synthesizes new minority class instances to improve the learning ability of machine learning algorithms on the minority class. SMOTE conducts the instance synthesis by means of interpolation between near neighbors. Initially, each instance in the dataset is represented as a feature vector. SMOTE starts by randomly selecting a minority class instance i from the dataset. It determines the k nearest neighbors of i from the remaining instances in the minority class by calculating their distance (e.g., the Euler distance) based on their feature vectors. SMOTE synthesizes new instances using n instances randomly selected from the k neighbors. The selection is random to increase the diversity of the generated new instances. For each selected instance, SMOTE computes a "difference vector" that represents the difference of the feature vectors between the selected instance and instance i. SMOTE synthesizes new instances by adding an offset to the feature vector of instance i, where the offset is the product of the difference vector with a random number between 0 and 1. SMOTE stops generating new instances until a predefined condition is satisfied (e.g., the ratio of instances in the majority and minority classes is the same).

	f_2: **Monthly revenue**	f_3: **Target**
i1	39	Optional
i2	42	Optional
i3	25	Optional
i4	100	Required
i5	150	Required
i6	200	Required
i7	400	Required

$\xrightarrow{\text{SMOTE}}$

	f_2: **Monthly revenue**	f_3: **Target**
i1	39	Optional
i2	42	Optional
i3	25	Optional
i4	100	Required
i5	150	Required
i6	200	Required
i7	400	Required
i8	40	Optional

Fig. 3. An Example of SMOTE Interpolation

(A) Model Building Phase, (B) Form Filling Relaxation Phase, and (C) Threshold Determination Phase
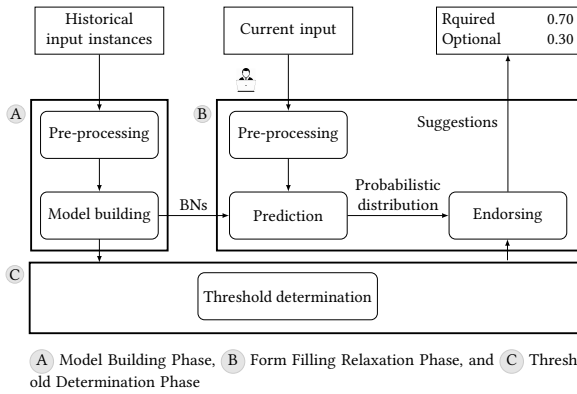
Fig. 4. Main Steps of the LACQUER Approach

Figure 3 illustrates the application of SMOTE to create new minority class instances. As shown in the table on the right, instances i_1, i_2, and i_3 belong to the minority class "Optional" of our target field. As a preliminary step, SMOTE computes the Euclidean distance between all the minority instances: $d(i_1, i_2) = \sqrt{(39 - 42)^2} = 3$, $d(i_1, i_3) = \sqrt{(39 - 25)^2} = 14$, and $d(i_2, i_3) = \sqrt{(42 - 25)^2} = 17$. SMOTE starts by randomly picking one instance from the minority class (e.g., i_2). Assuming that the value of k is equal to 1, SMOTE selects the nearest instance to i_2, which in our example is the instance i_1. In order to create a new instance i_8, SMOTE computes the *Difference vector* based on the feature vectors *Monthly revenue*$_{i_2}$ and *Monthly revenue*$_{i_1}$, and multiplies it by a random value λ between 0 and 1. The value of the "Monthly revenue" column in the synthetically created instance i_8 is equal to *Monthly revenue*$_{i_2}$ + *Difference vector*. In our example, assuming that the value of λ is equal to 0.7, the new value of the "Monthly revenue" field for i_8 is equal to $42 + ((39 - 42) * 0.7) = 40$.

4 APPROACH

In this section, we present our machine learning approach for data entry form relaxation named LACQUER (Learning-bAsed Completeness reQUirEments Relaxation).

As shown in Figure 4, LACQUER includes three phases: *model building*, *form filling relaxation*, and *threshold determination*. LACQUER preprocesses the historical input instances related to a data entry form and identifies the *meaningless* values in them. The historical input instances are divided in two parts: historical input instances for training (training input instances) and historical input instances for tuning (tuning input instances) used for threshold determination. In the first

phase, LACQUER builds BN models on the preprocessed training input instances to represent the completeness requirement dependencies between form fields. This phase occurs offline before deploying LACQUER as a completeness requirement relaxation tool for data entry. The form filling relaxation phase occurs during the data entry session and assumes that all the models have been built. During this phase, given a target field, LACQUER selects the BN model related to the target from all the BN models and predicts the completeness requirement of the target, taking into account the values of the filled fields captured during the form filling process. To improve prediction accuracy, LACQUER includes an endorser module that seeks to only provide users with predictions whose confidence level is higher than a minimum threshold. The value of the threshold is automatically determined in the threshold determination phase.

LACQUER is inspired by our previous work on automated form filling [5]; the main differences between the two approaches are discussed in section 6.

4.1 Pre-processing

The first two phases of LACQUER include a preprocessing step to improve the quality of the data in historical input instances as well as the current input instance. As mentioned in section 2.1, data entry forms can contain fields that are not applicable to certain users; this is the main cause of the presence of *missing values* and *meaningless values* in historical input instances. *Missing values* occur when users skip filling an (optional) field during form filling. A *meaningless value* is defined as any value filled into a form field that can be accepted during the validation check but does not conform with the semantics of the field. For example, given a data entry form with a textual field "Tax ID", if a user fills "n/a" in this field, the value can be accepted during the submission of the instance[3]; however, it should be deemed meaningless since "n/a" does not represent an actual "Tax ID".

For missing values, we replace them with a dummy value "Optional" in the corresponding field. As for the meaningless values, we first create a dictionary containing possible meaningless values based on domain knowledge. This dictionary is used to match possible meaningless values in historical input instances; we replace the matched values with "Optional". The rationale for this strategy is that it is common practice, within an enterprise, to suggest data entry operators some specific keywords when a field is not applicable for them. For example, our industrial partner recommends users to fill such fields with special characters such as "@" and "$". The overarching intuition behind replacing missing values and meaningless values with "Optional" is that, when data entry operators skip filling a field (resulting in a missing value in the form) or put a meaningless value, it usually means that this field is not applicable in the current context.

After detecting missing values and meaningless values, we preprocess other filled values. For textual fields, we replace all valid values with a dummy value "Required", reflecting the fact that data entry operators deemed these fields to be applicable. After preprocessing, all values in textual fields are therefore either "Required" and "Optional" to help the model learn the completeness requirement based on this abstract presentation. Numerical fields can be important to decide the completeness requirement of other fields. For example, companies reaching a certain monthly revenue can have some specific required fields. For this reason, we apply data discretization to numerical fields to reduce the number of unique numeric values. Each numeric value is represented as an interval, which is determined using the widely used discretization method based on information gain analysis [7]. We do not preprocess categorical fields since they have a finite number of candidate values. We keep the original values of categorical fields since users who select the same category value may share common required information. At last, we delete all the fields that are consistently marked as

---

[3]We assume the validation check does not check for the well-formedness of the string corresponding to the Tax ID.

f_1: **Company name** *(Textual)*	f_2: **Monthly revenue** *(Numerical)*	f_3: **Company type** *(Categorical)*	f_4: **Field of activity** *(Categorical)*	f_5: **Tax ID** *(Textual)*
UCI$\rightarrow$ Required	20$\rightarrow$[20,22)	Large enterprise	Real estate	T190
KDL $\rightarrow$ Required	21$\rightarrow$[20,22)	Large enterprise	Manufacturing	T201
EoP$\rightarrow$ Required	@$\rightarrow$Optional	Large enterprise	Manufacturing	T200
UNI$\rightarrow$ Required	39$\rightarrow$[39,41)	NPO	Education	n/a$\rightarrow$Optional

Fig. 5. Example of Pre-processed Historical Input Instances

"Required" or "Optional", because such fields do not provide any discriminative knowledge to the model.

During the data entry session, similar preprocessing steps are applied. We skip values filled in fields that were removed in historical input instances. We replace values in textual fields with "Required" and "Optional", as described above. We also map numerical values onto intervals and keep values in categorical fields.

The historical input instances are then divided in two parts that will be used separately for training (training input instances) and for the threshold determination (tuning input instances).

Application to the running example. Figure 5 shows an example of historical input instances collected from the data entry form presented in Figure 1. During the preprocessing phase, LACQUER identifies meaningless values in different fields (e.g., "n/a" and "@") and replaces them by the dummy value *Optional*. For the remaining "meaningful" values, LACQUER replaces values in the textual field "Company name" to the dummy value *Required*; values in the field "Monthly revenue" are discretized into intervals. In addition to historical input instances, LACQUER also preprocesses the input instance filled during the data entry session. For example, as shown in Figure 1, a user fills values *Wish*, *20*, *NPO,* and *Education* in fields "Company name", "Monthly revenue", "Company type", and "Field of activity", respectively. LACQUER will replace the value filled in the field "Company name" to "Required", since it is a meaningful value. LACQUER also maps the value in the field "Monthly revenue" into the interval *[20, 22).*

4.2 Model Building

The *model building* phase aims to learn the completeness requirement dependencies between different fields from training input instances related to a data entry form.

During the data entry session, we consider the filled fields as features to predict the completeness requirement of the target field (i.e., optional or required). However, in our previous work [5] we have shown that in an extreme scenario, users could follow any arbitrary order to fill the form, resulting in a large set of feature-target combinations. For example, given a data entry form with n fields, when we consider one of the fields as the target, we can get a total number of up to $2^{n-1} - 1$ feature (i.e., filled fields) combinations. Based on the assumption of identical features and targets [14] to train and test a machine learning model, a model needs to be trained on each feature-target combination, which would lead to training an impractical large number of models.

To deal with this problem, we select BNs as the machine learning models to capture the completeness requirement dependencies between filled fields and the target field, without training models on specific combinations of fields. As already discussed in our previous work [5], the reason is that BNs can infer the value of a target field using only information in the filled fields and the related PGM (see section 3.1); BNs automatically deal with the missing conditionally independent variables (i.e., unfilled fields).

Algorithm 1: Model Building

Input: Set of preprocessed historical input instances $I^F(t)'_{train}$ for training
Output: Dictionary of probabilistic graphical models $\mathcal{M}$

1   $\mathcal{M} \leftarrow$ empty dictionary;
2   List of fields $\textit{fields} \leftarrow \textit{getFields}(I^F(t)'_{train})$;
3   **foreach** field $f_i \in \textit{fields}$ **do**
4   Temporary training set $\textit{train}_{f_i} \leftarrow \textit{getDataSetForField}(I^F(t)'_{train}, f_i)$;
5   Oversampled Training set $\textit{train}_{f_i}^{oversample} \leftarrow \textit{SMOTE}(\textit{train}_{f_i}, f_i)$;
6   Model $M_i \leftarrow \textit{trainBayesianNetwork}(\textit{train}_{f_i}^{oversample})$;
7      $\mathcal{M}[f_i] \leftarrow M_i$
8   **end**
9   **return** $\mathcal{M}$;

A

f_2: **Monthly revenue** (Numerical)	f_3: **Company type** (Categorical)	f_4: **Field of activity** (Categorical)	f_5: **Tax ID** (Textual)
[20, 22)	Large enterprise	Real estate	T190
[20, 22)	Large enterprise	Manufacturing	T201
Optional	Large enterprise	Manufacturing	T200
[39, 41)	NPO	Education	Optional

Preprocessed historical input instances

D

The BN model for "Tax ID"

↓   ↑

B

f_2: **Monthly revenue** (Numerical)	f_3: **Company type** (Categorical)	f_4: **Field of activity** (Categorical)	f_5: **Tax ID** (Textual)
[20, 22)	Large enterprise	Real estate	Required
[20, 22)	Large enterprise	Manufacturing	Required
Optional	Large enterprise	Manufacturing	Required
[39, 41)	NPO	Education	Optional

Temporary training set for target "Tax ID"

C

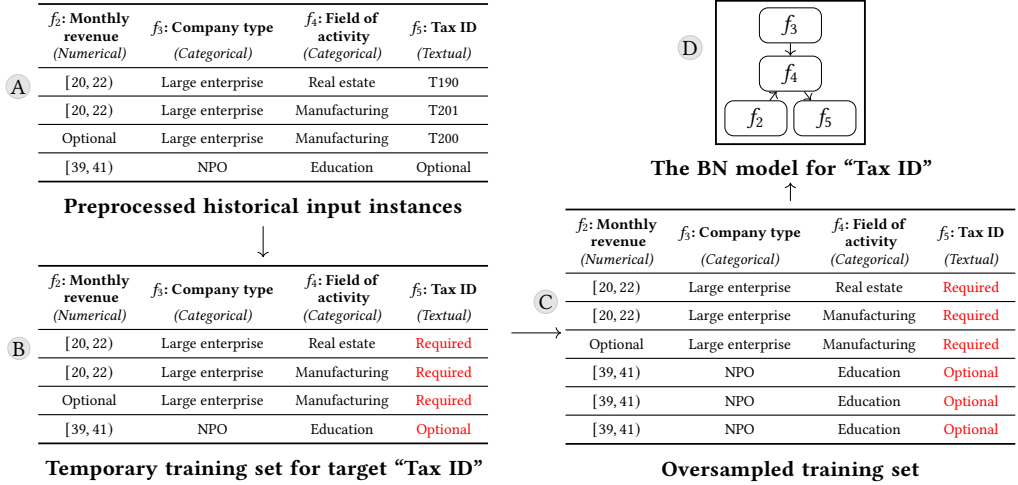| f_2: **Monthly revenue** (Numerical)	f_3: **Company type** (Categorical)	f_4: **Field of activity** (Categorical)	f_5: **Tax ID** (Textual)
[20, 22)	Large enterprise	Real estate	Required
[20, 22)	Large enterprise	Manufacturing	Required
Optional	Large enterprise	Manufacturing	Required
[39, 41)	NPO	Education	Optional
[39, 41)	NPO	Education	Optional
[39, 41)	NPO	Education	Optional

Oversampled training set

Fig. 6. Workflow of the Model Building Phase

In this work, LACQUER learns the BN structure representing the completeness requirement dependencies from training input instances. Each field in the data entry form represents a node (random variable) in the BN structure; the edges between different nodes are the dependencies between different fields. In order to construct the optimal network structure, BN performs a search-based optimization based on the conditional probabilities of the fields and a fitness function. As in our previous work [5], we use hill climbing as the optimizer to learn the BN structure with a fitness function based on BIC [40].

Algorithm 1 illustrates the main steps of this phase. LACQUER takes as input a set of preprocessed historical input instances $I^F(t)'_{train}$ for training and learning the completeness requirement dependencies (e.g., the input instances in block **A** of Figure 6). Initially, for each field f_i in the list of fields extracted from $I^F(t)'_{train}$ (line 2), we create a temporary training set where we consider the field f_i as the target (line 4). Since we aim to predict whether the target field is required or optional during form filling, in the temporary training set, we keep the value "Optional" in the target field f_i and label other values as "Required" (block **B** in Figure 6). These two values are the classes according to which to predict f_i.
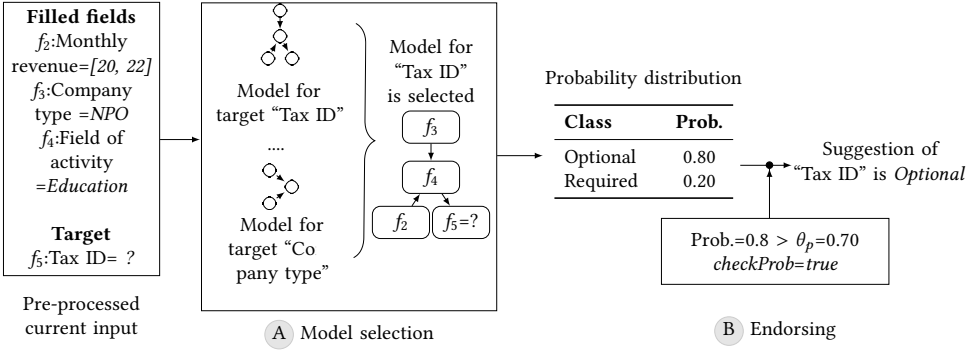
Fig. 7. Workflow for Form Relaxation Phase

However, we may not train effective classification models directly on this temporary training set. This is caused by the imbalanced nature of input instances for different classes. Users commonly enter correct and meaningful values during form filling. They only fill meaningless values in certain cases. As a result, the number of input instances having meaningless values (i.e., in the "Optional" class) is usually smaller than the number of input instances in the "Required" class. This can make the learning process inaccurate [29], since machine learning models may consider the minority class as noise [43]. The trained models could also over-classify the majority class due to its increased prior probability [29]. For example in block Ⓑ of Figure 6, considering that the column "f_5: Tax ID" is the current target, the number of instances in class "Required" is three, which is higher than the single instance in class "Optional". If we train a model on such imbalanced dataset, it might be difficult to learn the conditions (or dependencies) to relax this field as optional due to the small number of "Optional" instances.

To solve this problem, we apply SMOTE (line 5) on the temporary training set $train_{f_i}$ to generate an oversampled training set $train_{f_i}^{oversample}$ (as shown in block Ⓒ in Figure 6). After oversampling, both classes have the same number of input instances. We train a BN model M_i based on the oversampled training set for the target field f_i (line 6). For example, block Ⓓ in Figure 6 represents the model built for the target field "Tax ID". Following this step, we can obtain a BN model for each field. We save all the BN models in the dictionary $\mathcal{M}$ (line 7), where the key represents the name of the field and the value is the corresponding trained BN model. The output of Algorithm 1 is the dictionary $\mathcal{M}$.

Application to the running example. Given the preprocessed training input instances shown in block Ⓐ in Figure 6, LACQUER creates a temporary training set for each target (e.g., the field "Tax ID"), where LACQUER replaces the meaningful and meaningless values of the target field to *Required* and *Optional*, respectively (in block Ⓑ). The temporary training set is oversampled using SMOTE to create a balanced training set where the number of instances of both Required and Optional classes is the same (block Ⓒ of Figure 6). This oversampled training set is used to train a BN model for the target field "Tax ID". An example of the trained BN model is presented in block Ⓓ in Figure 6. After the model building phase, LACQUER outputs a model for each target. For the example of training input instances related to Figure 1, LACQUER returns five distinct models where each model captures the completeness requirement dependencies for a given target.

Algorithm 2: Form Filling Relaxation

Input: Dictionary of Models $\mathcal{M} = \{f_1 : M_1, \ldots, f_k : M_k\}$
   Set of Filled fields $C_t = \{\langle f_1^c, v_1^c \rangle, \ldots, \langle f_m^c, v_m^c \rangle\}$
   Target field f_p
   Threshold θ_p

Output: A Boolean $checkOP_p$, representing the decision to set field f_p to optional

1 Set $C_t' \leftarrow getPreprocessedData(C_t)$;
2 Model $M_p \leftarrow \mathcal{M}[f_p]$;
3 List of pairs of completeness requirements and probability distribution $\langle cr, pr \rangle_{list} = predictCR(M_p, C_t', f_p)$;
4 Top-ranked pair $\langle cr_{top}, pr_{top} \rangle = getTopRanked(\langle cr, pr \rangle_{list})$;
5 Boolean $checkOP_p \leftarrow isOptional(cr_{top})$;
6 **if** $checkOP_p$ **then**
7 | Boolean $checkProb \leftarrow (pr_{top} < \theta_p)$;
8 | **if** $checkProb$ **then**
9 | | $checkOP_p \leftarrow$ "false";
10 | **end**
11 **end**
12 **return** $checkOP_p$;

4.3 Form Filling Relaxation

The form filling relaxation phase is an online phase that occurs during the data entry session. In this phase, LACQUER selects the model $M_p \in \mathcal{M}$ corresponding to the target field f_p. This model is then used to predict the completeness requirement of the target f_p based on the filled fields C_t. The main steps are shown in Algorithm 2.

The inputs of the algorithm are the dictionary of trained models $\mathcal{M}$, the set C_t representing the filled fields during the entry session and their values, the target field f_p, and the endorsing threshold θ_p for f_p. The algorithm starts by applying the preprocessing techniques outlined in § 4.1 to the set of the filled fields in C_t in order to obtain a new set of preprocessed filled fields C_t' (line 1). LACQUER then selects the model M_p from $\mathcal{M}$ (line 2), since this model is trained for the target field f_p based on the oversampled data with a balanced number of instances for each class of f_p. With the selected model, LACQUER predicts the completeness requirement for f_p (line 3) and gets the top-ranked completeness requirement based on the prediction probability (line 4).

Endorsing. During the data entry session, values in filled fields do not always provide enough knowledge for the model to accurately predict the completeness requirement of a given target field. This happens because when training BN models, there may not be enough information in the training input instances to learn the dependencies among some fields with specific values.

However, in the context of form filling relaxation, it is important to provide accurate completeness requirement suggestions. On the one hand, wrongly predicting optional fields as required adds more constraints to the data entry form; users will be obliged to fill fields with meaningless values. On the other hand, wrongly predicting a required field as optional can result in missing information. In order to prevent this situation, LACQUER includes a heuristic-based endorser module that decides if the predicted completeness requirement is correct or not. Since our main goal is to relax the completeness requirement by predicting when a required field should be optional, we mainly use the endorser to endorse the prediction where the target field is predicted as "Optional". If the prediction is endorsed, we set the field to "Optional"; otherwise, we use its previous setting ("Required").

Specifically, LACQUER checks if the top-ranked completeness requirement is equal to "Optional"; it saves the result in the Boolean flag $checkOP_p$ (line 5). If the value of $checkOP_p$ evaluates to

---

Algorithm 3: Endorser Threshold Determination

    Input: Set of pre-processed historical input instances $I^F(t)'_{tune}$ for tuning
               Dictionary of Models $\mathcal{M} = \{f_0 : M_0, f_1 : M_1, \ldots, f_k : M_k\}$
    Output: Dictionary of thresholds θ

1 $\theta \leftarrow$ empty dict;
2 List of fields *fields* $\leftarrow getFields(I^F(t)'_{tune})$;
3 **foreach** *field* $f_i \in$ *fields* **do**
4 | $temp_{th} \leftarrow$ empty dictionary;
5 | $I^F(t)'_{tune_i} = getDataSetForField(I^F(t)'_{tune}, f_i)$;
6 | Model $M_i \leftarrow \mathcal{M}[f_i]$;
7 | **for** *n= 0 to 1 (step 0.05)* **do**
8 | | $predictedCRAll = predictCRAllInstances(M_i, I^F(t)'_{tune_i}, n)$;
9 | | $score = evaluate(I^F(t)'_{tune_i}, predictedCRAll)$;
10 | | $temp_{th}[n] = score$;
11 | **end**
12 | $\theta[i] = getBestScore(temp_{th})$;
13 **end**
14 **return** θ;

---

true, LACQUER analyses the probability distribution of the predicted completeness requirement of the target field since it reflects whether LACQUER has enough "confidence" in the prediction based on current information. We check if the probability for the field to be "Optional" is lower than a threshold θ_p for target f_p (line 7), saving the result in the Boolean flag *checkProb*. If the value of *checkProb* evaluates to *true*, we change the value of the Boolean flag *checkOP$_p$* to *false* (line 9) since it implies the model does not have enough "confidence" for variable inference and prediction; otherwise, LACQUER keeps the prediction as "Optional". The threshold θ is automatically determined; its value differs for different targets (as discussed in § 4.4). We use different threshold values because the prediction is done by models trained on different data and the variance of the data can have a significant effect on prediction accuracy [53].

Application to the running example. Figure 7 shows the process of predicting the completeness requirement of the field "Tax ID" based on the input values in Figure 1. LACQUER first selects the model related to the current target for prediction (block Ⓐ in Figure 7). Let us assume that, based on the BN variable inference, LACQUER predicts that field "Tax ID" has a probability of 0.80 to be Optional. Since the top predicted value is Optional, LACQUER activates the endorser module (block Ⓑ in Figure 7) to decide whether the level of confidence is acceptable. For example, let us assume the automatically decided threshold value for field "Tax ID" is 0.70 (i.e., θ_{taxID}=0.70). Since the probability value of the "Optional" class (0.80) is higher than this threshold, the Boolean flag *checkOP$_{TaxID}$* remains true. LACQUER decides to set the field "Tax ID" to Optional.

4.4 Endorser Threshold Determination

We automatically determine the value of the threshold in the endorser module for each target. This step occurs offline and assumes that the models in $\mathcal{M}$ built during the model building phase are available. The threshold θ_i for the target field i is determined with the set of preprocessed tuning input instances. The basic idea is that for each historical input instance in this subset, we consider all fields except field i to be filled and use the model trained for field i to predict its completeness requirement with different values of θ_i. We determine the value of θ_i based on the value that achieves the highest prediction accuracy on tuning input instances.

Table 1. Information about the Fields in the Datasets

Dataset	# of fields	# of instances	# of required fields	Name of required fields (% of missing and meaningless values)
NCBI	26	235538	6	sample-name(0), tissue(0.130), isolate(0.351), sex(0.351) biomaterial-provider(0.1), age(0.543)
PEIS	33	73082	19	legal name(0), contact name(0), first name(0.113), place of birth(0.127), native country(0.127), status(0), year of study(0.94), function(0), employer name(0.35), name of school/university(0.84), type of contract(0), contract start date(0.668), date of end of contract(0.974), field of activity (0), code NACE(0.123), primary activity(0), country of activity(0),percentage of activity(0)

The main steps are shown in Algorithm 3. The algorithm takes as input the set of preprocessed historical input instances for tuning $I^F(t)'_{tune}$ and the trained models $\mathcal{M}$. For each field f_i in the list of fields extracted from $I^F(t)'_{tune}$ (line 2), we generate a temporary dataset $I^F(t)'_{tune_i}$ where the value of field f_i is transformed into "Optional" and "Required" using the method presented in Figure 6(B) (line 5). We select the model corresponding to f_i from $\mathcal{M}$ (line 6) and use the selected model to predict the completeness requirement of field f_i based on the values of other fields in $I^F(t)'_{tune_i}$ (line 8). While predicting, we try different thresholds, varying from 0 to 1 with a step equal to 0.05. For each threshold value, we compare the predicted completeness requirement with the actual completeness requirement of field f_i in each input instance of $I^F(t)'_{tune_i}$ to calculate the prediction accuracy (line 9). LACQUER selects the value of θ_i that achieves the highest prediction accuracy value in $I^F(t)'_{tune_i}$ as the threshold for f_i (line 12). The algorithm ends by returning a dictionary containing the thresholds of all fields.

5 EVALUATION

In this section, we report on the evaluation of our approach for automated completeness requirement relaxation. First, we assess the overall accuracy of LACQUER when predicting the completeness requirement of fields in data entry forms, and compare it with state-of-the-art baselines. We then assess the performance of LACQUER, in terms of training time and prediction time, for practical applications. Last, we perform an ablation study to evaluate how the use of SMOTE (in the model building phase) and the heuristic-based endorser (in the form filling relaxation phase) affects the accuracy of LACQUER.

More specifically, we evaluated LACQUER by answering the following research questions (RQs):

RQ1 *Can LACQUER provide accurate predictions for completeness requirement relaxation, and how does it compare with baseline algorithms?*

RQ2 *Is the performance of LACQUER, in terms of training and prediction time, suitable for practical applications?*

RQ3 *What is the impact of using SMOTE and the heuristic-based endorser on the effectiveness of LACQUER?*

5.1 Dataset and Settings

Datasets. We selected the datasets used for the evaluation of LACQUER according to the following criteria: (1) data should be collected from a real data entry form; (2) the form fields should have different completeness requirements (i.e., required and optional); and (3) the data entry form should have obsolete required fields, where users could use meaningless values to pass the validation checks.

We identified two datasets meeting these criteria: one publicly available in the biomedical domain (dubbed NCBI) and another proprietary dataset, extracted from a production-grade enterprise information system, provided by our industrial partner (dubbed PEIS). Each dataset consists of data collected from *one* real-world data entry form.

Other datasets used in related work on adaptive data entry forms (see also section 6) were either not mentioned [19, 50], unavailable [6], or confidential [49]. In addition, we also analyzed datasets from surveys conducted in countries with transparency policies (e.g., the USA "National Survey on Drug Use and Health" [8]). However, these surveys do not contain a detailed specification defining the completeness requirement of each field and thus the corresponding dataset does not meet our selection criterion #2.

Both datasets are represented by a data table where each row corresponds to an input instance filled by a user and each column represents a specific field in the data entry form; an input instance represents all the field values as submitted by a user.

The NCBI dataset is composed of metadata for diverse types of biological samples from multiple species [4]; it has been used in previous work on automated form filling [5, 33]. This dataset provides the design of the corresponding data entry form in the CEDAR workbench [24] with the list of completeness requirements for different fields. Following the evaluation methodology described in previous work [33], we considered a specific subset from the NCBI dataset related to the species "Homo sapiens" for evaluation. We downloaded the dataset from the official NCBI website[4].

As shown in Table 1, the NCBI dataset contains 235 538 instances[5] and has 26 fields, six of which are required. These six fields are always required and are not subject to any additional conditions. We identify the meaningless values in the required fields using the strategy presented in section 4.1, i.e., mapping the actual value in the data with the dictionary of meaningless values obtained from the domain knowledge. In Table 1, next to each field we indicate the ratio of instances having missing or meaningless values. The ratio of meaningless values[6] varies from 0.1 (for *biomaterial-provider*) to 0.543 (for *age*). The case when the ratio of meaningless values is equal to 0 (i.e., *sample-name*) represents the situation where the field was correctly filled for all the instances in the dataset.

Based on the ratio of meaningless values in Table 1, we find that the number of instances for meaningless and valid values is imbalanced for most of the fields. For example, the ratio of meaningless values for *tissue* is 0.130. The field age has more meaningless values with a ratio of 0.543. The reason for this relatively high ratio could be that the completeness requirement (i.e., "Required") of this field does not conform with the actual need in the real world; that is, the field *age* is not required when the actual concept of "age" does not apply to a certain type of biomaterial (e.g., for protein TM-1 [47]).

The PEIS dataset contains the data filled through the web-based data entry form during the process of creating a new customer account. The dataset was extracted from the database of our industrial partner. Similar to the NCBI dataset, each row in the table represents an instance and each column represents a form field. We identified the mapping between column names in the table and field names in the data entry form using the available software documentation.

As shown in Table 1, the PEIS dataset has 33 fields, 19 of which are required (including conditionally required). In this dataset, nine of the required fields do not have missing/meaningless values (i.e., the ratio of meaningless values is 0). For the rest of the fields, the ratio of instances

---

[4] https://ftp.ncbi.nlm.nih.gov/biosample/

[5] The number of instances is different from that indicated in our previous work [5] since the preprocessing step in that work retained only instances with at least three fields being filled. In contrast, in this work we keep fields with missing values to analyze completeness requirements.

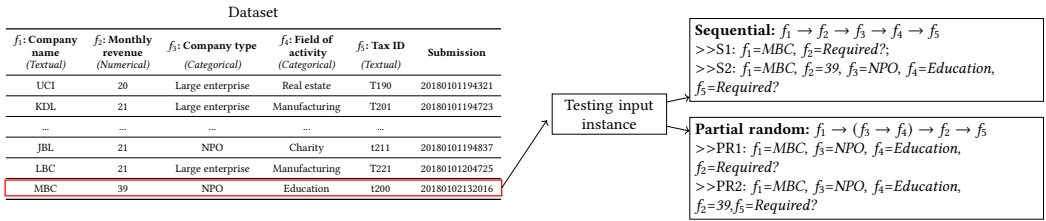[6] Required fields in the NCBI dataset have no missing values since they are always required.

f_1: Company name *(Textual)*	f_2: Monthly revenue *(Numerical)*	f_3: Company type *(Categorical)*	f_4: Field of activity *(Categorical)*	f_5: Tax ID *(Textual)*	Submission
UCI	20	Large enterprise	Real estate	T190	20180101194321
KDL	21	Large enterprise	Manufacturing	T201	20180101194723
...	...	...	...	...	...
JBL	21	NPO	Charity	t211	20180101194837
LBC	21	Large enterprise	Manufacturing	T221	20180101204725
MBC	39	NPO	Education	t200	20180102132016

Testing input instance

Sequential: $f_1 \rightarrow f_2 \rightarrow f_3 \rightarrow f_4 \rightarrow f_5$
>>S1: f_1=*MBC*, f_2=*Required?*;
>>S2: f_1=*MBC*, f_2=*39*, f_3=*NPO*, f_4=*Education*, f_5=*Required?*

Partial random: $f_1 \rightarrow (f_3 \rightarrow f_4) \rightarrow f_2 \rightarrow f_5$
>>PR1: f_1=*MBC*, f_3=*NPO*, f_4=*Education*, f_2=*Required?*
>>PR2: f_1=*MBC*, f_3=*NPO*, f_4=*Education*, f_2=*39*,f_5=*Required?*

Fig. 8. Example of Filling Orders

with missing or meaningless values ranges from 0.113 to 0.974. The reason behind having a high ratio of meaningless values in some fields, is that those fields are conditionally required. They are rarely to be required in real scenarios, which leads to many missing values.

Dataset Preparation. For the two datasets, we consider all the required fields as targets since we aim to learn the conditions to relax them as optional (for avoiding meaningless values and improving the overall data quality). However, we do not consider fields where the ratio of missing and meaningless values is 0, as they have no relaxation conditions to learn. We split the dataset into three subsets containing 80%, 10%, and 10% of input instances based on their submission time, used respectively for training, tuning, and testing. The input instances (excluding submission time) in the training set are used to train LACQUER. The validation set is used to decide the endorser threshold for each field following the strategy explained in section 4.4.

As for the testing input instances, since there is no information on the actual form filling order, we simulated two form filling orders for data entry, including "sequential filling" and "partial random filling".

The former corresponds to filling data entry forms in the default order, as determined by the form *tab sequence*, e.g., the navigation order determined by the HTML attribute `tabindex` in web UI designs [18]. It simulates the logical order many users follow to fill out forms, especially when they use a keyboard to navigate form fields [35]. The latter represents the case when designers group some semantics-related fields together and add controls to force users filling a group of fields sequentially [10]; outside these groups, users can fill fields randomly.

We simulated partial random filling by randomly generating a field order for each testing input instance while respecting the sequential order of the fields in the same group. In the case where there is no grouping or controls in the form, the partial random filling scenario turns into to a (complete) random filling scenario. The reason to simulate the partial random filling scenario is that by capturing the fields' grouping information, this scenario is more realistic compared to a (complete) random filling scenario.

In both form filling scenarios, the filled fields considered by LACQUER are the fields that precede each target. For each target field, we labeled as "Optional" the instances in which the target field contains missing or meaningless values; otherwise they are labeled as "Required". "Optional" and "Required" are the two classes that we consider as ground truth.

Dataset Preparation - Application Example. Figure 8 illustrates an example of application of our dataset preparation method. The table on the left-hand side of the picture represents the information submitted during the data entry session through the data entry form introduced in our motivating example in section 2.3. We split this dataset into a training set (80% of instances), a tuning set (10% of instances), and a testing set (10% of instances); let us assume the last row in the table is an instance in the testing set. The testing set is then processed to simulate the two form filling scenarios. The sequential filling scenario uses the filling order following the `tabindex` value of

the form fields. Assuming the `tabindex` order for the example is $f_1 \rightarrow f_2 \rightarrow f_3 \rightarrow f_4 \rightarrow f_5$, we can generate two test instances S1 and S2 (shown in the top right box of Figure 8) to predict the completeness requirement of f_2 and f_5, respectively. As for the partial random filling scenario, this scenario takes into account the controls or grouping of fields specified by the designer. For example, let us assume that "f_3 : company type" and "f_4: field of activity" belong to the same group of fields named "Business activities": this means that f_3 and f_4 should be filled sequentially. A possible filling order, randomly generated taking into account this constraint is then $f_1 \rightarrow (f_3 \rightarrow f_4) \rightarrow f_2 \rightarrow f_5$. The bottom right box in the figure shows the corresponding generated test instances PR1 and PR2.

Implementation and Settings. We implemented LACQUER as a Python program. We performed the experiments on the NCBI dataset with a computer running macOS 10.15.5 with a 2.30 GHz Intel Core i9 processor with 32 GB memory. As for the experiments on the PEIS dataset[7], we performed them on a server running CentOS 7.8 on a 2.60 GHz Intel Xeon E5-2690 processor with 125 GB memory.

5.2 Effectiveness (RQ1)

To answer RQ1, we assessed the accuracy of LACQUER in predicting the correct completeness requirement for each target field in the dataset. To the best of our knowledge, there are no implementations of techniques for automatically relaxing completeness requirements; therefore, we selected as baselines two rule-based algorithms that can be used to solve the form filling completeness requirements relaxation problem: *association rule mining* (ARM) [33] and *repeated incremental pruning to produce error reduction* (Ripper); these rule-based algorithms can provide form filling relaxation suggestions under different filling orders. ARM mines association rules having the format "if *antecedent* then *consequent*" with a minimal level of support and confidence, where the antecedent includes the values of certain fields and the consequent shows the completeness requirement of a target field for a given antecedent. ARM matches the filled fields with the antecedents of mined association rules, and suggests the consequent of the matched rules. Ripper is a propositional rule-based classification algorithm [11]; it creates a rule set by progressively adding rules to an empty set until all the positive instances are covered [31]. Ripper includes also a pruning phase to remove rules leading to bad classification performance. Ripper has been used in a variety of classification tasks in software engineering [23, 48]. Similar to ARM, Ripper suggests the consequent of the matched rule to users.

Methodology. We used Precision (*Prec*), Recall (*Rec*), Negative Predictive Value (*NPV*), and Specificity (*Spec*) to assess the accuracy of different algorithms. These metrics can be computed from a confusion matrix that classifies the prediction results into true positive (TP), false positive (FP), true negative (TN), and false negative (FN). In our context, TP means that a field is correctly predicted as required, FP means that a field is misclassified as required, TN means that a field is correctly predicted as optional, and FN means that a field is misclassified as optional. Based on the confusion matrix, we have $Prec = \frac{TP}{TP+FP}$, $Rec = \frac{TP}{TP+FN}$, $NPV = \frac{TN}{TN+FN}$, and $Spec = \frac{TN}{TN+FP}$. Precision is the ratio of correctly predicted required fields over all the fields predicted as required. Recall is the ratio of correctly predicted required fields over the number of actual required fields. NPV represents the ratio of correctly predicted optional fields over all the fields predicted as optional. Finally, specificity represents the ratio of correctly predicted optional fields over the number of actual optional fields.

We chose these metrics because they can evaluate the ability of an algorithm in predicting both required fields (using precision and recall) and optional fields (using NPV and specificity). A high

---

[7]Due to the data protection policy of our partner, we were obliged to run the experiments on the PEIS dataset using an on-premise, dedicated server that, however, could not be used to store external data (like the NCBI dataset).

Table 2. Effectiveness for Form Filling Relaxation

	Alg.	Sequential				Partial Random				Train	Predict (ms)	
		Prec	Rec	NPV	Spec	Prec	Rec	NPV	Spec	(s)	avg	min−max
NCBI	Ripper	0.63	0.79	0.17	0.20	0.69	0.83	0.25	0.16	349.29	0.18	0.18−0.19
	ARM	0.75	0.98	0.81	0.16	0.82	0.86	0.39	0.28	11.98	5.06	3−12
	LACQUER	0.76	0.98	0.91	0.20	0.84	0.98	0.76	0.37	145.98	75.83	33−144
PEIS	Ripper	0.66	0.73	0.60	0.29	0.58	0.62	0.84	0.56	240.37	0.24	0.15−0.54
	ARM	0.72	0.80	0.24	0.24	0.72	0.80	0.25	0.25	153.78	1.59	2−20
	LACQUER	0.88	0.98	0.72	0.62	0.90	0.97	0.75	0.64	1210.70	307	179−839

value of precision and recall means that an algorithm can correctly predict most of required fields (i.e., the positive class); hence, we can avoid business loss caused by missing information. A high value of NPV and specificity means that an algorithm can correctly predict most of the optional fields (i.e., the negative class); users will have fewer unnecessary constraints during form filling. In other words, we can avoid users filling meaningless values which may affect the data quality.

In our application scenario, we aim to successfully relax a set of obsolete required fields to "optional", while keeping the real required fields. Therefore, LACQUER needs to get high precision and recall values, which can preserve most of real required fields to avoid business loss. Meanwhile, the NPV value should be high, which means LACQUER can correctly avoid users filling meaningless values by relaxing the completeness requirements. Concerning the specificity, a relatively low value is still useful. For instances, a specificity value of 50% means LACQUER can reduce by half the data quality issues caused by meaningless values.

In the case of ARM, we set the minimum acceptable support and confidence to 5 and 0.3, respectively, as done in previous work [5, 33] in which it was applied in the context of form filling.

Results. Table 2 shows the accuracy of the various algorithms for the two form filling scenarios. LACQUER substantially outperforms Ripper in terms of precision and recall scores (i.e., columns *Prec* and *Rec*) for both sequential filling and partial random filling scenarios in both datasets (ranging from +13 pp to +32 pp in terms of precision score and from +15 pp to +35 pp in terms of recall score). When we compare LACQUER with ARM, they have similar results in terms of precision and recall scores on the NCBI dataset; however, LACQUER performs much better than ARM on the PEIS dataset (by at least +16 pp in terms of precision score and +17 pp in terms of recall score).

When looking at the *NPV* and specificity scores, on the NCBI dataset LACQUER and Ripper have the same specificity value for sequential filling; however, LACQUER can provide more accurate suggestions since it outperforms Ripper in terms of NPV score with an improvement of +74 pp. Concerning the partial random filling scenario on the NCBI dataset, LACQUER outperforms Ripper by +51 pp and +21 pp in terms of NPV and specificity scores, respectively. On the same dataset, when comparing LACQUER with ARM, the results shows that LACQUER always outperforms ARM for both form filling scenarios from +10 pp to +37 pp in terms of NPV score and from +4 pp to +9 pp in terms of specificity score. As for the PEIS dataset, for sequential filling LACQUER substantially outperforms the two baselines from +12 pp to 48 pp in terms of NPV score and from +33 pp to +38 pp in terms of specificity score. For partial random filling, Ripper achieves the highest NPV score, outperforming LACQUER by +9 pp; however, LACQUER outperforms both baselines in terms of specificity score by +8 pp to +39 pp.

Looking at the specificity score when applying LACQUER on PEIS and NCBI datasets, we can notice a difference ranging from +27 pp to +42 pp. This difference means that LACQUER can find more optional values in the PEIS dataset than in the NCBI dataset. We believe the main reason behind this difference is the quality of the training set. We recall PEIS is a proprietary dataset from

the banking domain. Data entry operators in the bank follow corporate guidelines for recommended values to be used when a field is not applicable, e.g., special characters like '@' or '$' (see section 4.1), resulting in higher quality data than the NCBI dataset. The latter, in fact, is a public dataset where anyone can submit data using the corresponding data entry form. Users do not follow any rule to insert special values when a field is not applicable. For this reason, the endorser module of LACQUER tends to remove more likely inaccurate suggestions, predicting only optional fields with high confidence. This explains the high value of NPV in the NCBI dataset, which is +19 pp higher than that in the PEIS dataset for the sequential filling scenario and +1 pp higher for the random filling scenario.

We applied Fisher's exact test with a level of significance $\alpha = 0.05$ to assess the statistical significance of differences between LACQUER and the baselines. The null hypothesis is that there is no significant difference between the prediction results of LACQUER and a baseline algorithm on the test instances. Given the output of each algorithm on the test instances we used during our evaluation, We created contingency tables summarising the decisions of LACQUER vs ARM and LACQUER vs Ripper for each form-filling scenario. Each contingency table represents the relationship between LACQUER and the other baseline in terms of frequency counts of the possible outputs (0: "Optional" and 1: "Required"). In other words, the contingency table counts the number of times both algorithms provide the same prediction (i.e., both predict a test instance as 0 or 1), and the number of times they have different prediction outputs (i.e., one algorithm predicts as 1 but the other predicts 0, and vice versa). These contingency tables are then used by Fisher's exact test to compute the p-value in order to reject or accept the null hypothesis. The result of the statistical test shows that LACQUER always achieves a significantly higher number of correct predictions than the baselines for the two form-filling scenarios on both datasets (*p-value* < 0.05).

These results have to be interpreted with respect to the usage scenario of a form filling relaxation tool. Incorrect suggestions can affect the use of data entry forms and the quality of input data. The *NPV* and specificity values achieved by LACQUER show that its suggestions can help users accurately relax the completeness requirement by 20 % to 64 % of the fields in data entry forms. Meanwhile, LACQUER can correctly preserve most ($\geq$ 97%) of the required fields required to be filled to avoid missing information (as indicated by the high precision and recall scores).

The answer to RQ1 is that LACQUER performs significantly better than the baseline algorithms. LACQUER can correctly relax at least 20% of required fields (with an NPV value above 0.72), while preserving the completeness constraints on most of the truly required fields (with a recall value over 0.98 and precision over 0.76).

5.3 Performance (RQ2)

To answer RQ2, we measured the time needed to perform the training and predict the completeness requirement of target fields. The training time evaluates the ability of LACQUER to efficiently update its models when new input instances are added daily to the set of historical input instances. The prediction time evaluates the ability of LACQUER to timely suggests the completeness requirement during the data entry phase.

Methodology. We used the same baselines and form-filling scenarios used for RQ1. The training time represents the time needed to build BN models (for LACQUER) or learn association rules (for ARM and Ripper). The prediction time is the average time needed to provide suggestions for target fields. We deployed LACQUER and baselines locally to avoid the impact of the data transmission time when assessing the prediction time.

Results. The results are presented in columns *Train* and *Predict* in Table 2. Column *Train* represents the training time in seconds. Column *Predict* contains two sub-columns representing the average

time and the minimum/maximum time (in milliseconds) needed to make a prediction on one test instance.

As shown in Table 2, Ripper has the highest training time for the NCBI dataset with 349.29 s. The training time of LACQUER (145.98 s) is between that of Ripper (349.29 s) and ARM (11.98 s) on the NCBI dataset. For the PEIS dataset, the training time of Ripper and ARM is equal to 240.37 s and 153.78 s, respectively; the training time of LACQUER is the highest: 1210.70 s (less than 20 minutes).

In terms of prediction time, LACQUER takes longer than ARM and Ripper to predict the completeness requirement of a field. On average, LACQUER takes 75.83 ms and 307 ms on the NCBI and PEIS datasets, respectively. The prediction time of ARM and Ripper depends on the number of rules used for matching the filled fields: the smaller the number of rules the shorter the prediction time. For LACQUER, the prediction time depends mostly on the complexity of BNs used when predicting. Such complexity can be defined in terms of the number of nodes and the number of dependencies among the different nodes in the BNs.

Taking into account the usage of our approach, the results can be interpreted as follows. Since the training phase occurs *offline* and *periodically* to train different BN models, the training time of 1210.70 s is acceptable from a practical standpoint; it allows for the daily (or even hourly) training of LACQUER in contexts (like enterprise software) with thousands of entries every day. Since LACQUER needs to be used during data entry, a short prediction time is important to preserve the interactive nature of a form-filling relaxation tool. The prediction time of LACQUER is acceptable according to human-computer interaction principles [26], which prescribe a response time lower than 1 s for tools that provide users with a seamless interaction. In addition, this prediction time is also comparable to the one achieved by our previous work on automated form filling [5]. Hence, LACQUER can be suitable for deploying in real enterprise systems.

The answer to RQ2 is that the performance of LACQUER, with a training time per form below 20 minutes and a prediction time of at most 839 ms *per target field, is suitable for practical application in data-entry scenarios.*

5.4 Impact of SMOTE and Endorser (RQ3)

LACQUER is based on two main modules: (1) SMOTE oversampling module, which tries to solve the class imbalance problem by synthetically creating new minor class instances in the training set (section 4.2), and (2) the endorsing module, which implements a heuristic that aims to keep only the optional predicted instances with a certain level of confidence. To answer this RQ we assessed the impact of these two modules on the effectiveness of LACQUER.

Methodology. We compared the effectiveness of LACQUER with three variants representing all the possible configurations of LACQUER: LACQUER-S, LACQUER-E, and LACQUER-SE. LACQUER-S represents the configuration where the SMOTE oversampling module is disabled and LACQUER provides predictions based on the imbalanced training set. LACQUER-E denotes the configuration where the endorser module is disabled and LACQUER directly returns the predictions to the user without checking whether the predictions have the required confidence in predicting fields as optional. LACQUER-SE is the configuration where both modules are disabled; this variant corresponds to the case where we use a plain BN. The different configurations are shown in Table 3 under column *Module*, where the two sub-columns *S* and *E* refer to the two modules "**S**mote" and "**E**ndorser". We used symbols '✓' and '✗' to specify whether a variant includes or not a certain module. LACQUER was run in its vanilla version as well as the additional variants using the same settings and evaluation metrics as in RQ1.

Results. As shown in Table 3, both modules have an impact on the effectiveness of LACQUER. The SMOTE oversampling module improves the ability of BNs to identify more optional fields; it

Table 3. Effectiveness of LACQUER with Different Modules

ID	Module		NCBI								PEIS							
	S	E	Sequential				Partial Random				Sequential				Partial Random			
			Prec	Recall	NPV	Spec	Prec	Recall	NPV	Spec	Prec	Recall	NPV	Spec	Prec	Recall	NPV	Spec
LACQUER-SE	✗	✗	0.78	0.88	0.36	0.32	0.84	0.89	0.44	0.33	0.90	0.92	0.70	0.67	0.96	0.96	0.75	0.58
LACQUER-E	✓	✗	0.78	0.82	0.56	0.32	0.85	0.77	0.39	0.65	0.89	0.94	0.67	0.76	0.92	0.88	0.57	0.85
LACQUER-S	✗	✓	0.76	0.98	0.51	0.19	0.83	0.98	0.66	0.22	0.88	0.99	0.70	0.55	0.91	0.99	0.77	0.52
LACQUER	✓	✓	0.76	0.98	0.91	0.20	0.84	0.98	0.76	0.37	0.89	0.99	0.74	0.64	0.90	0.97	0.75	0.64

improves the specificity score of a plain BN by at least +9 pp on the two datasets (LACQUER-E vs LACQUER-SE), except for the sequential filling scenario in the NCBI dataset where the specificity score stays the same. The endorser module mainly removes inaccurate optional predictions and keeps them as required to prevent missing information. This module leads to an increase in the recall value compared to the plain BN (LACQUER-SE vs LACQUER-S); it increases by at least +9 pp for the NCBI dataset in both scenarios. The improvement is smaller for the PEIS dataset where the recall increases by +7 pp and +3 pp for sequential and random filling scenarios, respectively. The endorser module affects also specificity, which decreases by at most 13 pp for both datasets when the endorser is used. The reason behind such decrease is that the endorser module removes possibly inaccurate predictions.

Comparing the results of LACQUER (with both modules enabled) with a plain BN (i.e, LACQUER-SE) on the NCBI dataset, the former improves NPV by +55 pp (0.91 vs 0.36) for the sequential filling scenario and by +32 pp (0.76 vs 0.44) for the random filling scenario. Since the endorser module considers the non-endorsed instances as required, it also increases recall by +10 pp and +9 pp for sequential and random filling scenarios, respectively. For the PEIS dataset, we find a slight increase in NPV of +4 pp and an increase of +6 pp for recall with sequential filling. For the partial random filling scenario, we notice that both LACQUER and LACQUER-SE have similar results, except for a higher specificity value +6 pp and a lower precision value of −6 pp for LACQUER. This loss in precision is expected since LACQUER keeps the default completeness requirement (i.e., required) for an instance for which the prediction confidence is low (i.e., the probability is lower than a threshold in endorser). These instances may include some truly optional cases with low confidence in the prediction; hence considering them as optional may slightly reduce the precision value.

The answer to RQ3 is that the SMOTE oversampling module and the endorser module improve the effectiveness of LACQUER.

5.5 Threats to Validity

To increase the generalizability of our results, LACQUER should be further evaluated on different datasets from different domains. To partially mitigate this threat, we evaluated LACQUER on two datasets with different data quality: the PEIS dataset, which is proprietary and of high quality, and the NCBI dataset, which is public and was obtained from an environment with looser data quality controls.

The size of the pool of training sets is a common threat to all AI-based approaches. We do not expect this problem to be a strong limitation of LACQUER since it targets mainly enterprise software systems that can have thousands of entries per day.

Since LACQUER needs to be run online during the data entry session, it is important to ensure seamless interaction with users. In our experiments (section 5.3), LACQUER was deployed locally. The response time of its prediction complies with human-computer interaction standards. However, the prediction time depends on the deployment method (e.g., local deployment or cloud-based).

This is not necessarily a problem since different engineering methods can help reduce prediction time such as parallel computing and a cache for storing previous predictions.

5.6 Data Availability

The implementation of LACQUER, the NCBI dataset, and the scripts used for the evaluation are available at https://figshare.com/articles/software/LACQUER-replication-package/21731603; LACQUER is distributed under the MIT license. The PEIS dataset cannot be distributed due to an NDA.

6 RELATED WORK

In this section, we discuss the work related to our approach. First, we review the existing approaches dealing with adaptive forms. Next, we provide a detailed comparison between LACQUER and LAFF. We conclude the section by presenting some tangential works that use BN to solve software engineering problems.

6.1 Adaptive Forms

The approach proposed in this paper is mainly related to approaches that implement adaptive forms for producing context-sensitive form-based interfaces. These approaches progressively add (remove) fields to (from) the forms depending on the values that the user enters. They use *form specification languages* [19] or *form definition languages* [6] to allow form designers to describe the dynamically changing behaviour of form fields. Such a behavior is then implemented through dedicated graphical user interface programming languages (such as Tcl/Tk) [50] or through server-side validation [6]. The dynamic behaviour of a form has also been modeled using a declarative, business process-like notation (DCR - Dynamic Condition Response graph [49]), where nodes in the graph represent fields and edges show the dynamic relations among fields (e.g., guarded transitions); the process declarative description is then executed by a process execution engine that displays the form. However, all these works assume that designers already have a *complete and final* set of completeness requirements describing the adaptive behaviour of the form during the design phase, which can be expressed through (adaptive) form specification/definition languages or tools. In contrast, LACQUER can automatically learn the different completeness requirements from the historical input instances filled by users, without requiring any knowledge from the form designers.

Although some approaches [1, 16] try to automatically generate data entry forms based on the schema of the database tables linked to a form (e.g., using column name and primary keys), they can only generate some "static" rules for fields. For example, if a column is "not null" in the schema, they can set the corresponding field in the form as (always) required. In contrast, LACQUER aims to learn conditions from the data so that completeness requirements of form fields can be *automatically and dynamically* relaxed during new data entry sessions.

6.2 Comparing LACQUER with LAFF

The overall architecture (including the use of the endorser module) of LACQUER has been inspired by LAFF, a recent approach for automated form filling of data entry forms [5]. In this subsection, we explain the similarities and differences between the two approaches.

Similarities between LAFF and LACQUER. Both LAFF and LACQUER are approaches that can be used during the form filling process. The main similarities between these approaches derive from the main challenges of form filling, i.e., dealing with (1) an arbitrary filling order and (2) partially filled forms.

The first challenge arises from the fact that users can fill a data entry form following an arbitrary order. Therefore, the filled fields (i.e., the features in our ML models) and the target field keep changing, leading to a large number of feature-target combinations. To avoid training a separate machine learning model on each feature-target combination, in this work, we are inspired by LAFF and use BNs to mine the relationships between filled fields and the target field.

As for the second challenge, LAFF addresses it using an endorser module. The main idea of the endorser module is to avoid providing inaccurate suggestions to the user when the form does not contain enough information for the model. Avoiding inaccurate suggestions is important for both approaches to gain the trust of users; for example, wrongly determining to relax a required field by making it optional may lead to missing information, thus hindering data completeness. For this reason, the second similarity between LAFF and LACQUER is the use of an endorser module.

Differences between LAFF and LACQUER. Table 4 shows the main differences between LACQUER and LAFF in terms of *goal*, *challenges*, *preprocessing*, *model building*, and *prediction*.

The main *goal* of LACQUER is to determine the completeness requirements of form fields. In contrast, LAFF provides form-filling suggestions for the values to be filled in categorical fields. Concerning the *challenges*, in addition to the shared ones discussed above, the relaxing completeness requirement problem has its own challenge when the dataset is highly imbalanced. We addressed this challenge in LACQUER by applying SMOTE.

The *preprocessing* step of the two approaches is completely different. Specifically, LAFF removes all textual fields from the data. In contrast, LACQUER transforms the values in textual fields into binary values. After the preprocessing, textual fields can only have one of two values: "Required" and "Optional". Moreover, the preprocessing step of LACQUER identifies meaningless values and replaces the matched values in the data with the value "Optional" (see section 4.1).

As for the *model building* phase, LAFF and LACQUER create a different set of BN models. LAFF creates $k + 1$ models, including a global model and k local models. The global model represents the BN created on the whole training data; the k local models are the BNs created based on the clusters of training data that share similar characteristics. The optimal number of clusters k is automatically determined with the elbow method. LACQUER creates n models where n represents the number of fields (targets) in the data entry form.

Finally, the differences regarding the *prediction* phase can be viewed from two perspectives: the type of targets and the endorser module. Concerning the target, LAFF only predicts possible values for categorical fields, no matter whether this field is optional or required. In contrast, LACQUER targets all types of required fields (e.g., textual, numerical, and categorical fields) to relax their completeness requirements. The endorser modules of LAFF and LACQUER differ as follows:

- The endorser module of LAFF endorses predictions based on two heuristics: the prediction confidence and the dependencies between the filled fields and the target. In contrast, the endorser of LACQUER is based only on the prediction confidence.
- LAFF uses a threshold to be determined manually, based on domain expertise, to endorse the prediction whereas LACQUER includes a phase to automatically determine the threshold for each target.

6.3 Using Bayesian Networks in Software Engineering Problems

Besides LAFF, BNs have been applied to different software engineering problems spanning over a wide range of software development phases, such as project management (e.g., to estimate the overall contribution that each new software feature to be implemented would bring to the company [34]), requirement engineering (e.g., to predict the requirement complexity in order to assess the effort

Table 4. Main differences between LAFF and LACQUER

		LAFF	LACQUER
Goal		• Providing form-filling suggestions for the values to be filled in categorical fields	• Determining the completeness requirements of form
Challenge		• Arbitrary filling order • Partial filling	• Arbitrary filling order • Partial filling • Highly imbalanced dataset
Preprocessing		• Textual fields are removed	• Values in textual fields are transformed into binary values ("Required" or "Optional") • Meaningless values are identified and replaced with the value "Optional"
Model building		• Creates $k+1$ models including a global model and k local models (one model for each cluster of data)	• Creates n models, one model for each field (target)
Prediction	Target	• Categorical field • LAFF can predict the value for both optional and required fields	• All textual, numerical, and categorical fields can be targets • Required field
	Endorser	• Use two heuristics based on prediction confidence and dependencies between filled fields and the target • The value of the threshold is manually decided based on domain expertise	• The endorser is based only on the prediction confidence • The value of the threshold is automatically determined during the threshold determination

needed to develop and test a requirement [44]), implementation (for code auto-completion [39]), quality assurance (e.g., for defect prediction [13, 28]), and software maintenance [41].

The main reason to use BN in software engineering (SE) problems is the ability of BN to address the challenges of dealing with "large volume datasets" and "incomplete data entries". First, software systems usually generate large amounts of data [41]. For instance, to improve software maintenance, companies need to analyze large amounts of software execution data (e.g., traces and logs) to identify unexpected behaviors such as performance degradation. To address this challenge, Rey Juárez et al. [41] used BN to build an analysis model on the data, since BN can deal with large datasets and high-dimensional data while keeping the model size small and the training time low. Second, incomplete data is a common problem in SE [15, 38]. For example, some metrics in defect prediction datasets might be missing for some software modules. To solve this challenge, Okutan and Yıldız [38] and Del Águila and Del Sagrado [15] used BN to train prediction models, because of its ability to perform inference with incomplete data entries. These two challenges confirm our choice of using BN to solve the relaxing completeness problem. Specifically, these two challenges are aligned with the challenges of form filling. During data entry sessions, a form is usually partially filled and LACQUER needs to provide decisions on incomplete data. Besides, in our context, we need to deal with large datasets since we mainly target enterprise software systems that can collect a huge number of entries every day.

7 DISCUSSION

7.1 Usefulness

The main goal of LACQUER is to prevent the entering of meaningless values by relaxing the data entry form completeness requirements. In order to assess the capability of LACQUER, we evaluated it with two real-world datasets, including a public dataset from the biomedical domain

and a proprietary dataset from the banking domain. These two datasets are related to existing data entry forms.

Experiment results show that LACQUER outperforms baselines in determining completeness requirements with a specificity score of at least 0.20 and a NPV score higher than 0.72. In the context of completeness requirement relaxation, these results mean that LACQUER can correctly (i.e., NPV ≥ 0.72) prevent the filling at least 20% meaningless values. In addition, LACQUER can correctly determine (with precision above 0.76) when a field should be required with a recall value of at least 0.97. This recall value means that LACQUER can almost determine all the required fields. The high precision value shows that LACQUER rarely incorrectly predicts optional fields as required. In other words, LACQUER will not add much extra burden to users by adding more restrictions during the form filling process.

As discussed in section 5.2, LACQUER can determine more optional fields (i.e., a higher specificity) in the PEIS dataset than in the NCBI dataset due to the higher data quality of the former. Since we target data entry functionalities in enterprise software, we expect to find similar conditions in other contexts in which data entry operators follow corporate guidelines for selecting appropriate values that should be filled when a field is not applicable. In such contexts, LACQUER is expected to provide results that are similar to those achieved on the PEIS dataset.

7.2 Practical Implications

This subsection discusses the practical implications of LACQUER for different stakeholders: software developers, end-users, and researchers.

7.2.1 Software Developers. LACQUER can help developers refactor data entry forms, which typically have many historical input instances and obsolete completeness requirements. LACQUER does not require developers to define a complete set of rules regarding the completeness requirement of form fields. Developers can integrate LACQUER into a given data entry form as an independent tool. Deploying LACQUER into a data entry form requires providing a mapping between a data entry form, and field names and column names in the dataset. The mapping needs only to be provided once and can be easily identified from Object Relational Mapping (ORM) and software design documentation. In addition to the mapping, deploying LACQUER requires a dictionary of meaningless values, i.e., the values that should be used during the data entry process when a field is not applicable. We expect this dictionary to be found in the user manual of the data entry software or in corporate guidelines, as it was the case for the PEIS dataset.

7.2.2 End Users. During the form filling process, obsolete required fields in the data entry form can affect the data accuracy since users have to enter meaningless values to skip filling these obsolete fields. LACQUER can automatically decide when a field should be required or not based on the filled fields and historical input instances. Our experiments show that LACQUER can correctly determine between 20% and 64% of optional fields, which reduces the user effort and the time taken during the form filling process.

7.2.3 Researchers. In order to avoid predicting required field as optional, LACQUER includes an endorser module to decide if the prediction is accurate enough to be provided to the user. We propose a novel strategy to automatically determine the threshold used in the endorser module. Hence, our endorser module does not require any configuration from the domain expert. We believe that such an endorser module can be adopted by other researchers in other recommender systems.

Fig. 9. Use case to combine LACQUER and LAFF together during form filling

7.3 Combining LACQUER with LAFF

Despite the differences explained in section 6, LACQUER and LAFF are complementary in practice. Both approaches can be combined as an AI-based assistant for form filling to help users fill forms and ensure better data quality.

Figure 9 shows a possible scenario that uses both approaches together during a form-filling session. In this example, we assume that the user follows the sequential filling order. First, after filling in the company name field, LACQUER can already check whether the "monthly income" field is required or not. Since "monthly income" is a numerical field, LAFF cannot perform a prediction (LAFF only supports categorial fields). In this example, LACQUER determines that the field is required, hence the user should fill it out. The "Company type" and "Field of activity" fields are both categorical. For these two fields, based on the filled fields, first LACQUER determines the completeness requirement for each field. Once the user clicks on a field, LAFF is enabled to provide a ranked list of possible values that can be used for this field. If the decision of LACQUER on a field is optional, LAFF can still be activated to provide suggestions as long as the user wants to fill in the field. Finally, let us assume that the "Tax ID" field (a numerical one) is optional by design. In this case, both LAFF and LACQUER are not enabled, since there is no need for LACQUER to relax a completeness requirement and the field is numerical and thus not compatible with LAFF.

8 CONCLUSION

In this paper we proposed LACQUER, an approach to automatically relax the completeness requirement of data entry forms by deciding when a field should be optional based on the filled fields and historical input instances. LACQUER applies Bayesian Networks on an oversampled data set (using SMOTE) to learn the completeness requirement dependencies between fields. Moreover, LACQUER uses a heuristic-based endorser module to ensure that it only provides accurate suggestions.

We evaluated LACQUER on two datasets, one proprietary dataset from the banking domain and one public dataset from the biomedical domain. Our results show that LACQUER can correctly determine 20% to 64% of optional fields and determine almost all the required fields (with a recall value of 0.97). LACQUER takes at most 839 ms to provide a suggestion, which complies with human-computer interaction principles to ensure a seamless interaction with users.

As a part of future work, we plan to conduct a user study to analyze the effect of LACQUER in reducing the meaningless values and the effort spent by users during the form filling process. We plan also to add an automated module that can detect meaningless values entered by the users during form filling, when such values have not been specified by the form designer. Furthermore, we plan to integrate LACQUER into platforms for the design of data entry forms [25, 36, 42] to help designers perform form refactoring. These platforms currently rely on rules defined by designers to specify completeness requirements during the design phase. LACQUER can be used to relieve designers from the task of defining such rules, since it only requires to indicate the required fields; during form filling, LACQUER will automatically suggest the completeness requirement of the required fields. LACQUER can also be extended to support sophisticated input fields that can handle multiple selections such us drop-down menus and checkbox groups. Finally, we plan to extend LACQUER to support updates of existing data entries as well as to determine whether fields previously marked as optional should become required.

Acknowledgements

REFERENCES

[1] Atia M Albhbah and Mick J Ridley. 2010. Using RuleML and database metadata for automatic generation of web forms. In *Proc.ISDA'10*. IEEE, IEEE, CAIRO, Egypt, 790–794.

[2] Maysoon Aldekhail and Djamal Ziani. 2017. Intelligent method for software requirement conflicts identification and removal: proposed framework and analysis. *International Journal of Computer Science and Network Security* 17, 12 (2017), 91–95.

[3] Alexander Avidan and Charles Weissman. 2012. Record completeness and data concordance in an anesthesia information management system using context-sensitive mandatory data-entry fields. *International Journal of Medical Informatics* 81, 3 (2012), 173–181.

[4] Tanya Barrett, Karen Clark, Robert Gevorgyan, Vyacheslav Gorelenkov, Eugene Gribov, Ilene Karsch-Mizrachi, Michael Kimelman, Kim D Pruitt, Sergei Resenchuk, Tatiana Tatusova, et al. 2012. BioProject and BioSample databases at NCBI: facilitating capture and organization of metadata. *Nucleic acids research* 40, D1 (2012), D57–D63.

[5] Hichem Belgacem, Xiaochen Li, Domenico Bianculli, and Lionel Briand. 2022. A Machine Learning Approach for Automated Filling of Categorical Fields in Data Entry Forms. *ACM Trans. Softw. Eng. Methodol.* 32, 2 (apr 2022), 40 pages. https://doi.org/10.1145/3533021 Just Accepted.

[6] Morten Bohøj, Niels Olof Bouvin, and Henrik Gammelmark. 2011. AdapForms: A framework for creating and validating adaptive forms. In *Proc. ICWE'11*. Springer, Springer Berlin Heidelberg, Berlin, Heidelberg, 105–120.

[7] Leo Breiman, Jerome Friedman, Charles J Stone, and Richard A Olshen. 1984. *Classification and regression trees*. CRC press, Boca Raton, Florida, USA.

[8] Brooklyn Lupari. 2015. national-survey-on-drug-use-and-health. https://catalog.data.gov/dataset/national-survey-on-drug-use-and-health-nsduh-2015. Accessed: 2023-05-30.

[9] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. 2002. SMOTE: synthetic minority over-sampling technique. *Journal of artificial intelligence research* 16 (2002), 321–357.

[10] Kuang Chen, Harr Chen, Neil Conway, Joseph M Hellerstein, and Tapan S Parikh. 2011. Usher: Improving data quality with dynamic forms. *IEEE Transactions on Knowledge and Data Engineering* 23, 8 (2011), 1138–1153.

[11] William W Cohen. 1995. Fast effective rule induction. In *Machine learning proceedings 1995*. Elsevier, San Francisco (CA), 115–123.

[12] Fabiano Dalpiaz, Ivor Van Der Schalk, Sjaak Brinkkemper, Fatma Başak Aydemir, and Garm Lucassen. 2019. Detecting terminological ambiguity in user stories: Tool and experimentation. *Information and Software Technology* 110 (2019), 3–16.

[13] Karel Dejaeger, Thomas Verbraken, and Bart Baesens. 2012. Toward comprehensible software fault prediction models using bayesian network classifiers. *IEEE Transactions on Software Engineering* 39, 2 (2012), 237–257.

[14] Ofer Dekel, Ohad Shamir, and Lin Xiao. 2010. Learning to classify with missing and corrupted features. *Machine learning* 81, 2 (2010), 149–178.

[15] Isabel M Del Águila and José Del Sagrado. 2016. Bayesian networks for enhancement of requirements engineering: a literature review. *Requirements engineering* 21 (2016), 461–480.

[16] A Elbibas and MJ Ridley. 2004. Developing Web entry forms Based on METADATA. In *International Workshop on Web Quality in conjunction with ICWE*. Citeseer, Trinity College Dublin, Dublin, 113–118 pages.

[17] Sergio Firmenich, Vincent Gaits, Silvia Gordillo, Gustavo Rossi, and Marco Winckler. 2012. Supporting users tasks with personal information management and web forms augmentation. In *Proc.ICWE'12*. Springer, Springer Berlin Heidelberg, Berlin, Heidelberg, 268–282.

[18] Susan Fowler and Victor Stanwick. 2004. *Web application design handbook: Best practices for web-based software.* Morgan Kaufmann, Amsterdam, Boston, USA.

[19] Martin R Frank and Pedro Szekely. 1998. Adaptive forms: an interaction technique for entering structured data. *Knowledge-Based Systems* 11, 1 (1998), 37–45.

[20] Nir Friedman, Dan Geiger, and Moises Goldszmidt. 1997. Bayesian network classifiers. *Machine learning* 29, 2-3 (1997), 131–163.

[21] José A Gámez, Juan L Mateo, and José M Puerta. 2011. Learning Bayesian networks by hill climbing: efficient methods based on progressive restriction of the neighborhood. *Data Mining and Knowledge Discovery* 22, 1-2 (2011), 106–148.

[22] C. Ghezzi. 2017. Of software and change. *Journal of Software: Evolution and Process* 29, 9 (2017), e1888. https://doi.org/10.1002/smr.1888 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/smr.1888 e1888 smr.1888.

[23] Baljinder Ghotra, Shane McIntosh, and Ahmed E Hassan. 2015. Revisiting the impact of classification techniques on the performance of defect prediction models. In *Proc.ICSE'15*, Vol. 1. IEEE, IEEE, Florence, Italy, 789–800.

[24] Rafael S. Gonçalves, Martin J. O'Connor, Marcos Martínez-Romero, Attila L. Egyedi, Debra Willrett, John Graybeal, and Mark A. Musen. 2017. The CEDAR workbench: an ontology-assisted environment for authoring metadata that describe scientific experiments. In *Proc. ISWC'17 (LNCS, Vol. 10588)*. Springer International Publishing, Cham, 103–110.

[25] Google LLC . 2016. Google Forms. https://docs.google.com/forms/. Accessed: 2021-12-09.

[26] Carrie Heeter. 2000. Interactivity in the context of designed experiences. *J. of Interactive Advertising* 1, 1 (2000), 3–14.

[27] Caroline Jarrett and Gerry Gaffney. 2009. *Forms that work: Designing Web forms for usability.* Morgan Kaufmann, Amsterdam, Boston, USA.

[28] Kawal Jeet, Nitin Bhatia, and Rajinder Singh Minhas. 2011. A bayesian network based approach for software defects prediction. *ACM SIGSOFT Software Engineering Notes* 36, 4 (2011), 1–5.

[29] Justin M Johnson and Taghi M Khoshgoftaar. 2019. Survey on deep learning with class imbalance. *Journal of Big Data* 6, 1 (2019), 1–54.

[30] Oksana Kulyk, Benjamin Maximilian Reinheimer, and Melanie Volkamer. 2017. Sharing Information with Web Services–A Mental Model Approach in the Context of Optional Information. In *Proc. HAS'17*. Springer, Springer, Cham, 675–690.

[31] Mario Linares-Vásquez, Collin McMillan, Denys Poshyvanyk, and Mark Grechanik. 2014. On using machine learning to automatically classify software applications into domain categories. *Empirical Software Engineering* 19, 3 (2014), 582–618.

[32] Ruchika Malhotra and Megha Khanna. 2017. An empirical study for software change prediction using imbalanced data. *Empirical Software Engineering* 22, 6 (2017), 2806–2851.

[33] Marcos Martínez-Romero, Martin J O'Connor, Attila L Egyedi, Debra Willrett, Josef Hardi, John Graybeal, and Mark A Musen. 2019. Using association rule mining and ontologies to generate metadata recommendations from multiple biomedical databases. *Database J. Biol. Databases Curation* 2019 (2019), 25 pages.

[34] Emilia Mendes, Mirko Perkusich, Vitor Freitas, and João Nunes. 2018. Using Bayesian Network to estimate the value of decisions within the context of Value-Based Software Engineering. In *Proc.EASE'18*. Association for Computing Machinery, New York, NY, USA, 90–100.

[35] Microsoft. 2013. Change the default tab order for controls on a form. https://support.microsoft.com/en-us/office/change-the-default-tab-order-for-controls-on-a-form-03d1599a-debf-4b66-a95b-e3e744210afe.

[36] Momentive Inc. 1999. Survey Monkey. https://www.surveymonkey.com/. Accessed: 2021-12-09.

[37] Kıvanç Muşlu, Yuriy Brun, and Alexandra Meliou. 2015. Preventing data errors with continuous testing. In *Proc.ISSTA'15*. Association for Computing Machinery, New York, NY, USA, 373–384.

[38] Ahmet Okutan and Olcay Taner Yıldız. 2014. Software defect prediction using Bayesian networks. *Empirical Software Engineering* 19 (2014), 154–181.

[39] Sebastian Proksch, Johannes Lerch, and Mira Mezini. 2015. Intelligent code completion with Bayesian networks. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 25, 1 (2015), 1–31.

[40] Adrian E Raftery. 1995. Bayesian model selection in social research. *Sociological Methodology* 25 (1995), 111–163.

[41] Santiago del Rey Juárez, Silverio Juan Martínez Fernández, and Antonio Salmerón Cerdán. 2023. Bayesian network analysis of software logs for data-driven software maintenance. *IET Software* 3, 17 (2023), 1–19.

[42] Rocketgenius Inc. 2007. Graviy Forms. https://www.gravityforms.com/. Accessed: 2021-12-09.

[43] Seyed Ehsan Roshan and Shahrokh Asadi. 2020. Improvement of Bagging performance for classification of imbalanced datasets using evolutionary multi-objective optimization. *Engineering Applications of Artificial Intelligence* 87 (2020), 103319.

[44] Halima Sadia, Syed Qamar Abbas, and Mohammad Faisal. 2022. A Bayesian Network-Based Software Requirement Complexity Prediction Model. In *Proc.ICCMDE'21*. Springer, Singapore, 197–213.

[45] Andrew Sears and Ying Zha. 2003. Data entry for mobile devices using soft keyboards: Understanding the effects of keyboard size and user tasks. *J. of Human-Computer Interaction* 16, 2 (2003), 163–184.

[46] Mirjam Seckler, Silvia Heinz, Javier A Bargas-Avila, Klaus Opwis, and Alexandre N Tuch. 2014. Designing usable web forms: empirical evaluation of web form improvement guidelines. In *Proc.CHI'14*. Association for Computing Machinery, New York, NY, USA, 1275–1284.

[47] Jiuling Song, Yonghe Zhou, Juren Zhang, and Kewei Zhang. 2017. Structural, expression and evolutionary analysis of the non-specific phospholipase C gene family in Gossypium hirsutum. *BMC genomics* 18, 1 (2017), 1–15.

[48] Qinbao Song, Yuchen Guo, and Martin Shepperd. 2018. A comprehensive investigation of the role of imbalanced learning for software defect prediction. *IEEE Transactions on Software Engineering* 45, 12 (2018), 1253–1269.

[49] Rasmus Strømsted, Hugo A López, Søren Debois, and Morten Marquard. 2018. Dynamic Evaluation Forms using Declarative Modeling. *BPM (Dissertation/Demos/Industry)* 2196 (2018), 172–179.

[50] Paul Thistlewaite and Steve Ball. 1996. Active forms. *Computer Networks and ISDN Systems* 28, 7-11 (1996), 1355–1364.

[51] Costas Vassilakis, Giorgos Laskaridis, Giorgos Lepouras, Stathis Rouvas, and Panagiotis Georgiadis. 2003. A framework for managing the lifecycle of transactional e-government services. *Telematics and Informatics* 20, 4 (2003), 315–329.

[52] Wan MN Wan-Kadir and Pericles Loucopoulos. 2004. Relating evolving business rules to software design. *Journal of Systems Architecture* 50, 7 (2004), 367–382.

[53] Zeqing Wu and Weishen Chu. 2021. Sampling strategy analysis of machine learning models for energy consumption prediction. In *Proc.SEGE'21*. IEEE, IEEE, Oshawa, ON, Canada, 77–81.

[54] Junwen Yang, Utsav Sethi, Cong Yan, Alvin Cheung, and Shan Lu. 2020. Managing data constraints in database-backed web applications. In *Proc.ICSE'20*. IEEE, Association for Computing Machinery, New York, NY, USA, 1098–1109.