DISSERTATION

Defense held on 15/09/2023 in Luxembourg

to obtain the degree of

DOCTEUR DE L'UNIVERSITÉ DU LUXEMBOURG

EN INFORMATIQUE

by

Hichem BELGACEM
Born on 28 May 1992 in Monastir (Tunisia)

ML-BASED DATA-ENTRY AUTOMATION AND DATA ANOMALY DETECTION TO SUPPORT DATA QUALITY ASSURANCE

Acknowledgments

I would like to acknowledge and give my warmest thanks to my supervisor Dr. Domenico Bianculli, for his significant effort to make my research work successful. His dedication, insightful advice, and scientific approach kept me constantly engaged with my research.

I would like to express my special gratitude to Prof. Lionel C. Briand, for his support and guidance to accomplish this thesis. I am honored to have had the opportunity to learn from his knowledge and academic excellence.

I owe a deep gratitude to my co-supervisor, Dr. Xiaochen Li for significantly enriching this thesis. His dynamism and unwavering generosity helped me accomplish my tasks.

Special thanks go to BGL BNP Paribas, our industrial partner, and especially to Anne Goujon, Michael Stanisiere, Fernand Lepage, Clément Lefebvre Renard, and Andrey Boytsov for their availability and for providing us with different datasets.

Special thanks go to my family and friends for their continuous support throughout my PhD.

Abstract

Data plays a central role in modern software systems, which are very often powered by machine learning (ML) and used in critical domains of our daily lives, such as finance, health, and transportation. However, the effectiveness of ML-intensive software applications highly depends on the quality of the data. Data quality is affected by data anomalies; data entry errors are one of the main sources of anomalies. The goal of this thesis is to develop approaches to ensure data quality by preventing data entry errors during the form-filling process and by checking the offline data saved in databases.

The main contributions of this thesis are:

1. LAFF, an approach to automatically suggest possible values of categorical fields in data entry forms.
2. 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.
3. LAFF-AD, an approach to automatically detect data anomalies in categorical columns in offline datasets.

LAFF and LACQUER focus mainly on preventing data entry errors during the form-filling process. Both approaches can be integrated into data entry applications as efficient and effective strategies to assist the user during the form-filling process. LAFF-AD can be used offline on existing suspicious data to effectively detect anomalies in categorical data.

In addition, we performed an extensive evaluation of the three approaches, assessing their effectiveness and efficiency, using real-world datasets.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Context

Data plays a central role in modern software systems, which are very often powered by machine learning (ML) and used in critical domains of our daily lives, such as finance, health, and transportation [TLA$^+$22]. However, the effectiveness of ML-intensive software applications highly depends on the quality of the data [SE03]. When data quality is poor, the output of these systems cannot be trusted. This makes data quality of utmost importance since it impacts how trustworthy and reliable these applications are (e.g., decision support systems).

Data quality is affected by data anomaly. An anomaly is defined as data instances that deviate from other data instances [Haw80]. Data anomalies can arise in many practical situations, such as wrong values entered by users during the data entry process through form filling (e.g., typos) and errors made during data management (e.g., faulty sources of data) or data integration [MBM15].

Specifically, data entry is important to collect users' inputs through data entry forms [JG09, SZ03]. Yet, the data entry process is time consuming and error-prone. Users have to spend a lot of effort to make sure that the entered values are the right ones for each field, which makes the data entry process slow and frustrating [GC06]. This situation inevitably leads to a situation in which wrong values are provided as input for a field. As a result, many data quality problems are introduced when users fill the data entry form. Existing statistics reveal that data entry typically has an error rate of over 1% [FGJ08]. If such erroneous data is not detected and it is transferred into the software system, this erroneous data may propagate and affects all the other connected information. For example, erroneous input data in retailing systems alone leads to a waste of $2.5 billion each year for consumers [FGJ08]. Further, data entry errors in spreadsheets mislead business decisions, causing additional costs in correcting the errors [SR12]. Even worse, data entry has been a top cause of medication errors [Ame05], which resulted in at least 24 deaths in the US in 2003 [MBM15].

Data entry errors can be in the form of wrong values or meaningless values. *Wrong values* occur when the user wrongly fill the value of a field in the form. Wrong values can be filled in different type of fields in the form (e.g., textual, numerical, etc.) and especially in categorical fields. Categorical fields represent the fields that provide a list of candidate values (also called "options") from which the user has to choose (e.g., country). Such fields are likely to generate data quality issues in critical domains such as medical and financial domains. The reason is, in order to fill such fields, users need to spend tremendous effort to go through the list of options and compare between them to choose the right value. For example, empirical studies have shown that more than half (54.5%) of the data errors in a medical record system were caused by the candidate value selection error [QMR$^+$20]. Users could wrongly select the job role, the modality of care, or the drugs unintentionally [QMR$^+$20, KJ10], causing potential medical negligence.

Meaningless values are caused by the evolving nature of data entry forms. Data entry forms use completeness requirements to specify the fields that are required or optional to fill in for collecting necessary information from different types of users. However 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 can be represented by random symbols/strings that pass the validation checks (e.g, "@", "99") or by randomly selecting a value from a categorical field. They threaten the quality of the filled data, and also affect the trust of data engineers on the data saved in the database. Data engineers may not be able to differentiate between meaningless and correct values.

All the aforementioned data entry errors are considered as *data anomalies* when they are recorded into a database. *Data anomalies* must be detected;

once detected, they have to be fixed or excluded from the data used in ML-intensive applications. If the anomalies are not detected in a timely fashion, they may propagate and cause more data quality issues, leading to serious problems that affect decision-making [SE03]. For example, one study showed that the financial loss because of fraud, a kind of data anomaly in the financial domain, has increased by almost 56.6% since 2009. The value of this loss was estimated to be 5.1 trillion USD in 2018 [GB19]. Another study revealed that the average cost of network downtime resulting from data anomalies is 100 000 USD per hour; this value can grow as more people are using or become dependent on web applications [KLZ+19]. This explains the fact that anomaly detection has been applied in several critical domains such as fraud detection [POKB20], health care [CVM+21], and network intrusion identification in computer science [JK21].

1.2 Motivation

Different works tried to improve the data quality by preventing data entry errors (wrong values / meaningless values) during the form filling process and by checking the presence of data anomalies on existing data that have been filled before or merged from different sources of data.

1.2.1 Addressing the wrong value problem

To address the wrong value problem, different works have been proposed to help users filling fields, especially for categorical fields. For example, some form filling tools [Goo08, WZNN17] tried to recommend frequently selected values in a field or values selected by a user in some similar fields from 3rd-party software systems. However, the usage of these tools may violate enterprise security policies, since they rely on information from 3rd-party software systems. Moreover, these tools provide limited support due to the low accuracy of their suggestions [WGV+11].

Furthermore, existing automated form filling approaches exhibit some limitations when dealing with (1) forms filled following an arbitrary order, and (2) partially filled forms. The first situation arises because it is very common for data entry forms to have little or no restrictions on the order of fields in which users can enter data. Users are free to choose any field as their next target or even go back and edit fields they have already filled out. Some automated form filling approaches [HS94] need a defined form filling order before building a recommendation model; however, this assumption is unrealistic from a practical standpoint. As for the second situation, at a certain time during the data entry session, a form is usually partially filled, meaning that an approach for automated form filling can only use information in currently filled fields. However, when this is not sufficient to predict the target, existing approaches [MROE^{+}19, HS94] yield inaccurate suggestions. Based on these two situations it is important to have a form filling tool that can deal with different orders of filling with also taking into consideration of the limited information available in the form in the case of partially filled forms.

1.2.2 Addressing the meaningless values problem

To prevent entering meaningless values during the form filling process, existing works proposed adaptive form tools [FS98, BBG11, SLDM18], which give the opportunity to form designers to mark required fields as optional when certain conditions are met. These tools first require a *complete and final* set of completeness requirements representing the situations for which a field should be required or optional. Then, they use intermediate representations such as XML [BBG11] and dynamic condition response graphs [SLDM18] to represent the completeness requirements rules and implement adaptive behaviors. In addition, there are commercial tools (e.g., Gravity Forms [Roc], Google Forms [Goo]) 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 evolving nature of the software and the complexity of the domain (with hundreds of fields), identifying a comprehensive set of completeness requirements is not a practical solution. Moreover, even if they could be identified, such completeness requirements could become quickly obsolete, limiting the use of existing adaptive form tools. For these reason, it is important to have a solution that does not need predefined completeness requirements as input.

1.2.3 Detecting data anomalies

To detect data anomalies, many approaches have been proposed. However, their focus has been mainly on numerical data. The main idea is to define a proximity measure between data instances [IPM16], and use this measure to detect data instances that deviate from the majority of the data. In contrast, according to an existing study [TH19], only a fewer studies attempt to detect anomalies for categorical data. Since in practice data is often described by categorical attributes [IPM16], we focus in this thesis on detecting anomalies in categorical data.

There are two reasons to motivate this work. First, categorical data anomaly is more challenging to detect [IPM16]. The fundamental issue is to define a proximity measure over categorical values [RSMMA+19]; however, it is not easy to devise a criterion to separate between anomalous and non-anomalous categorical data [AFPS22]. Second, empirical studies show that more than 50% of anomalies in critical applications (e.g., medical records) are in categorical fields [QMR+20].

Although anomaly detection for categorical data has been investigated in the literature (e.g., with frequency-based approaches [PCC16, PTAJ16], clustering-based approaches [SMA12, SMA13], semi-supervised approaches [NBS12]), our preliminary experiments show that the effectiveness of these approaches is not stable across datasets. More specifically, especially since these approaches include different parameters to be tuned, one technique can perform greatly in terms of accuracy on one dataset but poorly on another.

1.3 Research Contributions

The overall goal of this dissertation is to propose approaches to ensure data quality by preventing data entry errors during the form filling process and by checking offline data saved in databases. Our goal is to develop approaches that deal with the different challenges and limitations of the state of the art discussed above. More specifically we identify the following research objectives:

(1) Developing an efficient automated form filling suggestion tool for categorical fields to help users filling categorical fields.

(2) Developing an efficient form filling relaxation tool to prevent filling meaningless values during the form filling process.

(3) Developing an efficient anomaly detection tool to detect inconsistent data in offline datasets.

To achieve the first research goal, we propose LAFF, a learning-based automated approach for filling categorical fields in data entry forms. LAFF first builds Bayesian Network models by learning field dependencies from a set of historical input instances, representing the values of the fields that have been filled in the past. To improve its learning ability, LAFF uses local modeling to effectively mine the local dependencies of fields in a cluster of input instances. During the form filling phase, LAFF uses such models to predict possible values of a target field, based on the values in the already-filled fields of the form and their dependencies; the predicted values (endorsed based on field dependencies and prediction confidence) are then provided to the end-user as a list of suggestions.

To achieve the second research goal, 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.

To achieve the third research goal, driven by the high accuracy of existing form filling tools such as LAFF, we propose a LAFF-based Anomaly Detection approach (“LAFF-AD” in short) to effectively detect categorical data anomalies. The basic idea of LAFF-AD is to take advantage of the learning ability of LAFF to perform value inference on suspicious data. LAFF-AD runs an adaptation of LAFF that handles offline prediction (i.e., not in real-time during the data entry process) to predict the value of a suspicious categorical field in the suspicious instance. LAFF-AD leverages the output of LAFF to detect data anomaly with a heuristic-based anomaly detection module.

1.4 Dissemination

The publications resulted from this theses are listed below.

Published papers

- Hichem Belgacem, Xiaochen Li, Domenico Bianculli, and Lionel Briand. A machine learning approach for automated filling of categorical fields in data entry forms. ACM Transactions on Software Engineering and Methodology, 2023, vol. 32, no 2, p. 1–40.
  This paper is the basis for Chapter 3; it presents our form filling approach for categorical fields.

- Hichem Belgacem, Xiaochen Li, Domenico Bianculli, and Lionel Briand. Learning-Based Relaxation of Completeness Requirements for Data Entry Forms. ACM Transactions on Software Engineering and Methodology, Just accepted
  This paper is is the basis for Chapter 4; it presents our completeness requirement relaxation approach.

Unpublished reports

- Hichem Belgacem, Xiaochen Li, Domenico Bianculli, and Lionel Briand. Automated anomaly detection for categorical data by repurposing form filling recommender systems.
  This paper is the basis for Chapter 5; it presents our anomaly detection approach for categorical fields in offline data.

1.5 Organization of the Thesis

Chapter 2 provides some fundamental background on machine learning algorithms we rely on, including *Bayesian networks*, *K-modes*, and *Synthetic Minority Oversampling Techniques (SMOTE)*.

Chapter 3 presents LAFF, our approach for filling categorical fields in data entry forms.

Chapter 4 presents LACQUER, our approach for relaxing the completeness requirements of data entry forms.

Chapter 5 presents LAFF-AD, our approach for detecting data anomalies for categorical columns in offline data.

Chapter 6 summarizes the thesis contributions and discusses perspectives on future work.

Chapter 2

Background

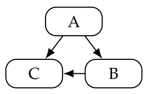
Before illustrating our approaches, we briefly introduce basic machine learning algorithms we rely on.

2.1 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 [FGG97] 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 [GMP11]) is applied to search the graphical structure. The search algorithm initializes a random structure, and then iteratively adds/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 [Raf95]) based on the nodes' conditional probabilities and on Bayes' theorem [FGG97]. Structure learning stops when it finds a graphical structure that minimizes the fitness function.



$P(A)$

a	$\bar{a}$
0.2	0.8

$P(B \mid A)$

A	B: b	B: $\bar{b}$
a	0.4	0.6
$\bar{a}$	0.1	0.9

$P(C \mid A, B)$

A	B	C: c	C: $\bar{c}$
a	b	0.9	0.1
a	$\bar{b}$	0.4	0.6
$\bar{a}$	b	0.4	0.6
$\bar{a}$	$\bar{b}$	0.1	0.9

Figure 2.1: An Example of BN and the Probability Functions of its Nodes

Figure 2.1 shows an example of the BN structure with three random variables: variable B depends on variable A; variable C depends on variables A and B. 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 2.1 each variable has two values; the probability table for B reflects the conditional probability $P(B \mid A)$ between A and B on these values.

Variable inference infers unobserved variables from the observed variables and the graphical structure of the BN using Bayes' theorem [FGG97]. For example, we can infer the probability of $C = c$ when the value of A is a (denoted as $P(c \mid a)$) as follows:

$$\begin{aligned}
P(c \mid a) &= \frac{P(a,c)}{P(a)} = \frac{P(a,b,c) + P(a,\bar{b},c)}{P(a)} \\
&= \frac{P(c \mid a,b)P(b \mid a)P(a) + P(c \mid a,\bar{b})P(\bar{b} \mid a)P(a)}{P(a)} \\
&= \frac{0.9 * 0.4 * 0.2 + 0.4 * 0.6 * 0.2}{0.2} = 0.6
\end{aligned}$$

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 [FGG97].

2.2 K-modes

K-modes is a clustering algorithm that extends the *k*-means one to enable clustering of categorical data [Hua98]. The algorithm first randomly selects *k* instances in the data set as the initial centroids. Each instance is represented with a vector of categorical attributes. The algorithm clusters the instances in the dataset by calculating the distances between the instances and each centroid. The distance, also called dissimilarity measure, is defined as the total mismatches of the corresponding categorical attributes of an instance and of a centroid. Based on the clustering results, new centroids are selected, which represent the modes of categorical attributes in each cluster. The algorithm then re-clusters the instances according to the new centroids. This process is repeated until the centroids remain unchanged or until it reaches a certain number of iterations.

2.3 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 [SGS18, MK17a].

To address this problem, many imbalanced learning approaches have been proposed in the literature. One of them is SMOTE [CBHK02]; 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).

Figure 2.2 illustrates an example of 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 mi-

	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

Figure 2.2: An Example of SMOTE Interpolation

nority 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 $42 + ((39 - 42) * 0.7) = 40$.

Chapter 3

A Machine Learning Approach for Automated Filling of Categorical Fields in Data Entry Forms

3.1 Overview

In this chapter, we propose *LAFF*, a Learning-based Automated Form Filling approach, for filling categorical fields. The basic idea of LAFF is to build machine learning models based on input instances (i.e., fields and the corresponding values provided in input) obtained from data entry forms that have been filled in the past (hereafter called *historical input instances*); such models represent dependencies among fields in historical input instances. Using these models, the already-filled fields in a data entry form can then be used as features to predict the possible values of a given target field. LAFF aims to be used by developers, who can integrate it into their data entry form implementations.

To deal with forms filled in an arbitrary order (which would result in a huge number of combinations of filled fields (features) and target fields to handle), LAFF utilizes Bayesian Networks (BN) to mine the dependencies between any field combinations, without assuming, a priori, an order for form filling. Moreover, to improve its learning ability, LAFF uses a local modeling strategy to cluster historical input instances; further, it builds additional local BNs, which learn fine-grained field dependencies from the clusters of historical input instances. These local models capture additional dependencies that might not have been captured by the model trained on the entire historical dataset. Once the trained models are available, LAFF uses them in the form filling suggestion phase, which occurs during the data entry session: given a target field, LAFF selects one of the available BNs and predicts the possible values of the field based on the values in the already-filled fields. To deal with partially filled forms (which might lead to inaccurate suggestions), LAFF includes a heuristic-based endorser, which determines whether the values predicted in the previous step are accurate enough to be returned to the user, based on the analysis of the field dependencies and of the predicted probability distribution of the values for the target field.

We evaluated LAFF using form filling records from both a public dataset and a proprietary dataset extracted from a production-grade enterprise information system in the banking domain. The experimental results show LAFF can yield a large number of accurate suggestions with a *Mean Reciprocal Rank* (*MRR*) value above 0.73 and a prediction coverage rate ranging from 0.70 to 0.87, significantly outperforming a state-of-the-art approach based on association rule mining by 11 pp to 27 pp (with pp = percentage points) in terms of *MRR* on both datasets. Furthermore, LAFF is efficient; it takes at most $317\,\mathrm{ms}$ to provide suggestions for input instances of the proprietary dataset.

To summarize, the main contributions of this chapter are:

- The LAFF approach, which addresses the problem of automated form filling for categorical fields, an important user interface challenge in many software systems. To the best of our knowledge, LAFF is the first work to combine BNs with local modeling and a heuristic-based endorser to provide accurate form filling suggestions, even for arbitrary filling orders and partially filled forms.

- An extensive evaluation assessing the effectiveness and efficiency of LAFF and comparing it with the state of the art.

The rest of the chapter is organized as follows. Section 3.2 provides a motivating example and explains the basic definitions of automated form filling and its challenges. Section 3.3 describes the different steps and the core algorithms of LAFF. Section 3.4 reports on the evaluation of LAFF. Section 3.5

discusses the usefulness, practical implications, and limitations of LAFF. Section 3.6 surveys related work. Section 3.7 concludes the chapter.

3.2 Data Entry Form Filling

In this section, we introduce the concepts related to data entry forms, provide a motivating example, define the problem of automated form filling, and discuss its challenges.

3.2.1 Types of Fields in Data Entry Forms

A data entry form is typically composed of many fields (also called input parameters [WZNN17] or elements [ABY16]), which can be of different types: textual, categorical, numerical, and file. Textual and numerical fields collect free text and numerical values respectively; users can freely input any value that is compliant with the field input validation rules. Categorical fields provide a list of candidate values from which the user has to choose (e.g., country); the source of candidate values is defined statically [ADK07]. File fields are used to upload files, such as images and videos. During software design, these fields are associated with specific UI widgets, based on the corresponding type. For example, developers can use a list box or combo box to collect categorical data [W3C17].

3.2.2 Motivating Examples

Although software users frequently fill forms, such activity is time-consuming and error-prone [RNDL+08]. In the following, we describe two examples illustrating the main challenges faced while filling data entry forms with categorical fields.

Example 1: *Users require focused attention to choose options from categorical fields*.

Alison is a student majoring in biology. She uses information management platforms (such as NCBI [BCG+12] and SAIKS [Gaf20]) to record the basic information of biological samples. Given a biological sample (e.g., the genus *Pratylenchus*), she needs to fill fields “sex”, “tail shape”, and “species name” of the sample in a data entry form from such a system. All three fields are categorical with predefined values. After filling the first two fields, Alison starts to select the species name. However, the genus *Pratylenchus* currently includes over 80 valid species [Gaf20]. She has to scroll down the list to check for the relevant species. Although she can search a species by inputting the first letter of the name, many similar species names are presented (e.g., *pratensisobrinus*, *pseudocoffeae*, *pseudofallax*, and *pseudopratensis*). She still requires focused attention to choose among these textually similar options in a limited time. According to existing studies, about half of the data entry errors are caused by selection error [QMR+20, WBL+13]

Example 2: *Users require cognitive effort to match options with the actual value they plan to fill in*. The second example is inspired by a use case of our industrial partner. The use case refers to the opening of a bank account for business customers; a simplified data entry form for this activity is shown on the left-hand side of Figure 3.1. We will use this data entry form as a running example to explain the form filling problem and illustrate our solution. For simplicity, let us assume that the data entry form contains only two fields: “company type” (textual) and “(company) primary field of activity” (categorical). When a company called *SmartLease* requests to open a bank account, the bank clerk Bill (who is the end-user interfacing with the data entry form) asks the *SmartLease* representative about the company type and inputs “leasing (company)” in the corresponding field; then, Bill selects “other financial services” for the “primary field of activity” field. Several weeks after the account opening, the data quality division of the bank detects a potential data quality issue regarding *SmartLease*, in the form of a mismatch between the actual activity implied by the company’s type and operations, and the company activity recorded when the account was open. This issue can be quickly solved by checking with *SmartLease* and amending the “primary field of activity” field with the correct value: “leasing services”. Nevertheless, such an issue can cause a business loss: for example, by knowing the actual “primary field of activity” of the company, the bank could have offered targeted products to its customer since the beginning of the business relation. Further investigation reveals that this issue occurred because, as a new employee in the bank, Bill was not familiar with all the 75 options defined in the “primary field of activity” field. He browsed the list for a limited time, compared the candidate values with the actual value he intended to fill in and finally selected an inappropriate value.

The aforementioned problems cannot be solved satisfactorily by existing solutions that support filling categorical fields, including those based on the search-by-keyword functionality, web browser plugins for autofill, as well as approaches that build field ontologies.

First, some solutions help users locate the candidate values in categorical fields with the search-by-keyword functionality. This functionality cannot solve the issues in the two examples, since users need to carefully compare textually similar values (as shown in Example 1) and have the burden to remember all the options to avoid searching an inappropriate value (as shown in Example 2).

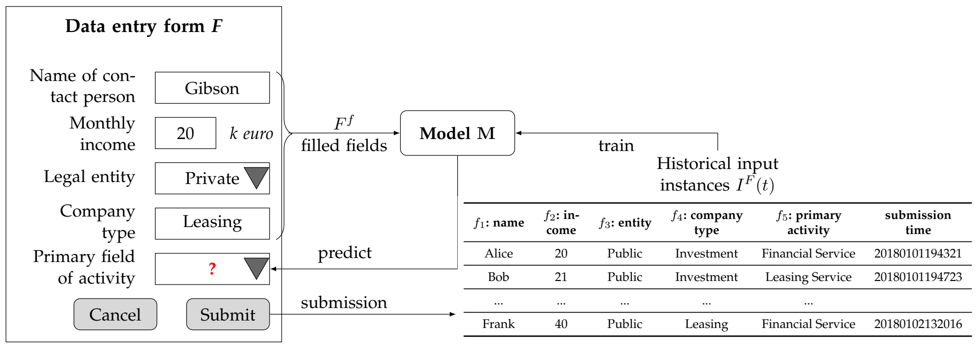
Second, web browser plugins such as Chrome Autofill Forms [Goo08] provide automatic form filling, but they simply reuse the inputs provided in past forms to automatically fill out fields in different forms with the same information (for example “zip code”). They do not leverage the knowledge provided by already filled fields to provide “intelligent” suggestions. Moreover, these tools are usually personalized for a single user, and cannot be used in the context of enterprise software systems, in which the end-user fills the same form with different information. As shown in Example 2, a bank clerk works daily with several bank accounts of different customers and cannot directly reuse the input instance of a customer to pre-fill an input form for another customer.

Third, some approaches automatically build ontologies for form filling [AHS12, WGV$^+$11, AGLH10]. They map a ‘target’ field (e.g., “zip code”) in a form to ‘source’ fields (e.g., “postal code” or “postcode”) in other filled forms to support data exchange across software systems. However, for domain-specific software systems (e.g., biology information management platforms), many fields are domain-specific (such as “tail shape” and “species name” in Example 1) and cannot be easily mapped to fields in other forms. In addition, due to legal or security policies, software systems for governments and enterprises have constraints on sharing records across systems (as the banking system in Example 2).

For all the above reasons, there is a need to design a semi-automated method that developers can adopt to support and guide users during the form filling activity.

3.2.3 Problem Definition

In this chapter, we deal with the automated form filling problem, which can be informally defined as the problem of suggesting possible values for the form fields a user is about to fill in, based on the values of the other fields and on the values provided as input in previous data entry sessions. We target *categorical fields* for automated filling since they require cognitive effort and focused attention for users to choose among the (typically) large set of options. The task of filling categorical fields may be slow and frustrating to users [GC06], and may lead to data quality issues, as shown in the examples above. We define the automated form filling problem as follows.



f_1: **name**	f_2: **income**	f_3: **entity**	f_4: **company type**	f_5: **primary activity**	**submission time**
Alice	20	Public	Investment	Financial Service	20180101194321
Bob	21	Public	Investment	Leasing Service	20180101194723
...	...	...	...	...	
Frank	40	Public	Leasing	Financial Service	20180102132016

Figure 3.1: The Automated Form Filling Problem

Let F be a data entry form with n fields $F = \{f_1, f_2, \ldots, f_n\}$; each field f_i can take values from a domain V_i (which always includes a special element $\perp$ representing an empty field); let $F^c \subseteq F$ be the set of categorical fields.

When a form F is being filled, at any time the fields can be partitioned in two groups: fields that have been already filled (denoted by F^f) and unfilled fields (denoted by F^u); we have that $F^f \cup F^u = F$ and $F^f \cap F^u = \emptyset$.

When a filled form F is about to be submitted (e.g., to be stored in a database), we define an *input instance* of F $I^F = \{\langle f_1, v_1\rangle, \ldots, \langle f_n, v_n\rangle\}$ with $f_i \in F$ and $v_i \in V_i$, as the set of pairs $\langle \mathit{field}, \mathit{value}\rangle$ from F; we use the subscript t_j as in $I^F_{t_j}$ 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 form F that have been submitted up to a certain time instant t; $I^F(t) = \{I^F_{t_i}, I^F_{t_j}, \ldots, I^F_{t_k}\}$, where $t_i < t_j < t_k < t$. Hereafter, we drop the superscript F when it is clear from the context.

The automated form filling problem can be defined as follows. Given a (partially filled) form $F = F^f \cup F^u$, a set of historical input instances $I^F(t)$, and a target field $f_p \in (F^u \cap F^c)$ to fill, we want to build a model M that at time t can predict a value v_p for f_p based on F^f and $I^F(t)$. Notice that in this problem definition the filling order of the fields in F is unrestricted.

3.2.3.1 Application to the running example

Figure 3.1 shows an example explaining the automated form filling problem. We have a data entry form F for a banking system with five fields: f_1:"name of contact person", f_2:"monthly income", f_3:"legal entity", f_4:"company type", and f_5:"primary field of activity". Among them, fields "legal entity" and "primary field of activity" are categorical (i.e., $F^c = \{f_3, f_5\}$). During the data entry session, users provide values for these fields, which are stored in a database upon submission of the form. The table on the right-hand side of Figure 3.1 shows some historical input instances filled by the bank customers through the data entry form; the submission timestamp t of these input instances is automatically recorded. In the table, each row represents an input instance (e.g., $I^F_{20180101194321} = \{\langle\textit{“name”}, \textit{Alice}\rangle, \ldots, \langle\textit{“primary activity”}, \textit{Financial Service}\rangle\}$); the column names correspond to the field names in the data entry form[1]. With these historical input instances, we can build a model M to learn the relationships of values filled in fields f_1 to f_5 by different customers. Notice the submission timestamp is not used for model building; it is only used for distinguishing different input instances.

Continuing the example, let us assume that, at a certain point of the data entry session, a customer has provided the values *Gibson*, *20*, *Private*, and *Leasing* for fields f_1 to f_4 respectively, as shown on the left-hand side of Figure 3.1; the unfilled field f_5 ("primary field of activity") is the next (categorical) field to fill in. Our goal is to use the model M to predict the value of f_5 based on the values of the filled fields f_1 to f_4.

3.2.4 Challenges of Automated Form Filling

Several automated form filling approaches have been proposed [MROE+19, HS94, TBA17]; the basic idea is to mine dependencies among fields from the values recorded in previous form filling sessions, to build recommendation models. These models can then be used to suggest possible values on a target field based on the filled fields in the current form. Nevertheless, state-of-the-art approaches exhibit some limitations when dealing with (1) forms filled following an arbitrary order, and (2) partially filled forms.

First, while filling in a data entry form, it is very frequent to have little or no restriction on the order of user' inputs. A user may select any field as the next target or even go back and modify already filled fields. In other words, the set of filled fields (F^f) and the target field to suggest ($f_p \in F^u$) keep changing. This scenario is different from the one considered by many recommender systems in the software engineering domain [MK17b, MBM+11], in which models are trained on predefined features/attributes (e.g., code metrics) to predict a specific target (e.g., source code defects). Some automated

[1]In the case of this illustrative example, we assume that the mapping between field names and column names can be retrieved in some way, for example by manually analyzing the existing software design documentation or software implementation. We provide more explanations on identifying such a mapping in section 3.5.2.1.

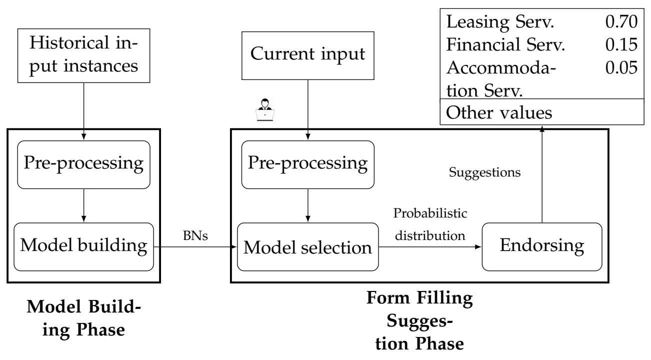


Figure 3.2: Main Steps of the LAFF Approach

form filling approaches [HS94] require a fixed form filling order before building the recommendation models. However, this assumption is unrealistic from a practical standpoint. Although some approaches [MROE+19, TBA17] are insensitive to the form filling order (e.g., suggesting possible values of a target field based on the frequency of values in historical inputs), they may not provide accurate suggestions due to their limitations in accurately mining dependencies among fields (as discussed in section 3.4.2). Hence, one of the challenges in automated form filling is how to build recommendation models (e.g., by mining dependencies among fields) without making any assumption on the order in which fields are filled.

Second, at a certain time during the data entry session, a form is usually partially filled: this means that a recommender system for automated form filling can only use the knowledge in currently filled fields (F^f). However, when the filled fields do not provide enough knowledge to predict the target—based on our preliminary experiments—existing approaches [MROE+19, HS94] yield inaccurate suggestions. The challenge, when dealing with partially filled forms, is how to discard low-confidence suggestions, in order to make suggestions only when a high degree of confidence is achieved.

3.3 Approach

In this section, we present our machine-learning based approach for form filling, named LAFF (Learning-based Automated Form Filling).

LAFF includes two phases: *model building* and *form filling suggestion*, whose main steps are shown in Figure 3.2. In the former, LAFF analyzes historical input instances of a data entry form and uses dependency analysis to build BNs that represent the fields in the form and their conditional dependencies.

	f_1: **name (Textual)**	f_2: **income (Num.)**	f_3: **entity (Categ.)**	f_4: **company type (Textual)**	f_5: **primary activity (Categ.)**
●	~~*Alice*~~	20→[20,22)	Public	Investment	Financial Service
●	~~Bob~~	21→[20,22)	Public	Investment	Leasing Service
■	~~*Carl*~~	39→[39,41)	Private	Investment	Leasing Service
■	~~*David*~~	39→[39,41)	Private	Leasing	Leasing Service
■	~~*Eliot*~~	40→[39,41)	Private	Leasing	Leasing Service
▲	~~*Frank*~~	40→[39,41)	Public	Leasing	Financial Service

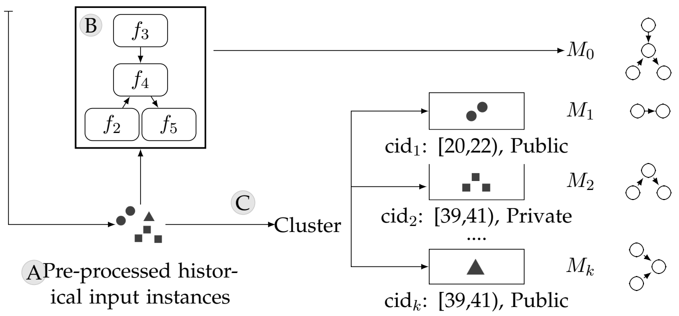


Figure 3.3: Example of Model Building on Pre-processed Historical Input Instances

This phase occurs offline, before deploying LAFF as an assistive technology for data entry. The *form filling suggestion* phase occurs during the data entry session: given a target field, LAFF selects a BN among those built in the *model building* phase and predicts possible values based on the values in the already-filled fields and their conditional dependencies; the predicted values and the corresponding predicted probability (endorsed based on field dependencies and prediction confidence) are then provided to the end-user as suggestions.

3.3.1 Pre-processing

Both phases of LAFF include a pre-processing step to improve the quality of the form filling data; this step is based on best practices for predictive data mining [AKV19].

Typically, historical input instances have many *missing values* due to the presence of optional fields in input forms. Fields that have a high number of missing values do not provide representative information for model building; hence, they can be removed. We remove fields for which $T_p^v\%$ or more of the values are missing, where T_p^v is a user-configurable threshold (with default value equal to 90).

We also remove file and textual fields that have a high number of unique values, since they typically correspond to form fields for which users frequently provide new string values (e.g., the textual field "name"). To identify such fields, we compute the ratio of unique values of a field in the historical input instances; if the ratio is larger than a user-configurable threshold T_p^u (with default value equal to 0.9), the corresponding field is removed.

Furthermore, we delete a historical input instance if more than $T_p^m\%$ of its field values are missing, where T_p^m is a user-configurable threshold (with default value equal to 50). After deletion, we perform data imputation [JQWX16] on the remaining data that exhibit missing values. Numerical fields are imputed using the mean value of this field; categorical and textual fields are imputed using a default label "UNKNOWN".

We also apply data discretization to numerical fields to reduce the number of unique values. Numerical values are transformed into discrete intervals based on information gain analysis, a widely used discretization method first proposed in decision trees [BFSO84].

During the data entry session, we ignore values in the fields that were removed in historical input instances, and map numbers onto intervals.

3.3.1.1 Application to the running example

The table at the top of Figure 3.3 shows an example of historical input instances filled through the data entry form in Figure 3.1. Each row is a historical input instance filled by a user. During pre-processing, LAFF removes the field "name" since all its values are unique (we crossed out the text of these values with a hatch pattern to represent the removal). Also, the values of field "income" are discretized into intervals. During the data entry session, as shown in Figure 3.1, a user fills the fields "name" with *Gibson*, "income" with *20*, "legal entity" with *Private*, and "company type" with "Leasing"; "primary activity" is the next field to be filled. Through the application of the pre-processing steps, LAFF ignores the value for field "name" and maps the value *20* of field "income" to the interval *[20, 22)*.

3.3.2 Model building

The goal of the *model building* phase is to mine dependencies from historical input instances of a data entry form.

Algorithm 1: Model Building

```
Input: Pre-processed historical input instances $I^F(t)'$
Output: List of probabilistic graphical models $\mathcal{M}$
        Historical input instance clusters $C$
1  $\mathcal{M} \leftarrow$empty list;
2  $M_0 \leftarrow trainBayesianNetwork(I^F(t)')$;
3  $\mathcal{M}.append(M_0)$;
4  independent field set $F^I \leftarrow \emptyset$;
5  foreach field $f_i \in M_0$ do
6      if $getParents(M_0, f_i) = \emptyset$ then
7          $F^I \leftarrow \{f_i\} \cup F^I$;
8      end
9  end
10 number of cluster $k \leftarrow elbowMethod(I^F(t)', F^I)$;
11 $C = \{I^F(t)'_1, \dots, I^F(t)'_k\} \leftarrow kModes(I^F(t)', F^I, k)$;
12 for $i \leftarrow 1$ to $k$ do
13     $M_i \leftarrow trainBayesianNetwork(I^F(t)'_i)$;
14     $\mathcal{M}.append(M_i)$;
15 end
16 return $\mathcal{M}$, $C$;
```

Due to the arbitrary order for filling the form, the filled fields and the target field keep changing. When we take the filled fields as features to predict the target field, the arbitrary form filling order results into a large set of feature-target combinations. For example, let us consider a data entry form with n fields, with $t \leq n$ of them being categorical and thus representing the possible targets. When we take one of the categorical fields as the target, assuming that a random order is used for form filling, in principle users may fill any subset of the remaining $n-1$ fields, resulting in a total of $2^{n-1}-1$ possible combinations of filled fields (i.e., features). The total number of feature-target combinations is equal to $t * (2^{n-1} - 1)$. Normally, a model would need to be trained on each target-features combination to ensure the assumption of identical features and target [DSX10] in the model building and form filling suggestion phases. As we will show through our evaluation in section 3.4, adopting such an approach would require to train more than 220 000 models on one of our datasets. The total time required to train this large number of models would be impractical for a production-grade system.

To solve this problem, we capture dependencies with BNs, in which variables correspond to form fields. By using BNs, we can analyze the dependency between filled fields and target fields without training models on specific combinations of features (i.e., filled fields) and target field. In addition, as mentioned in section 2.1, BNs are robust when dealing with missing values. This means that even when a data entry form is partially filled, BNs can still infer the probability distribution of target fields using only the information in the filled fields and the underlying PGM.

In this work, we learn the structure of BN from the pre-processed historical input instances. Following the workflow of BN presented in section 2.1, we represent each field in the historical input instances as a random variable. BN computes the conditional probability between any two fields and uses a search-based optimizer to automatically optimize the structure of BN based on the conditional probability of fields and the fitness function. In this study, we use hill climbing as the optimizer, because it shows a good trade-off between computational demands and the quality of the models learned [GMP11]. We define the fitness function in terms of BIC [Raf95], which aims at best fitting the data, while avoiding over-fitting by complex structures. The element denoted with B in Figure 3.3 shows an example of the BN structure learned from the data in block A (where the different black shapes correspond to the various rows in the table at the top of Figure 3.3).

Algorithm 1 illustrates the main steps of this phase. LAFF takes as input the pre-processed historical input instances $I^F(t)'$ as the training data to mine field dependencies. Initially, we learn the BN over the entire training data (line 2). This global model, denoted as M_0, represents the general dependencies among fields. However, historical input instances may form different groups that share similar characteristics. For example, in the historical input instances of Figure 3.3, users having the same value for fields "income" and "legal entity" may share specific values for fields "company type" and "primary activity". The global model M_0 may not learn the fine-grained field dependencies for specific values of "income" and "legal entity" due to the influence of input instances with other values for those fields. For example, using the entire dataset in Figure 3.3, one could determine that the conditional probability of having "primary activity" equal to *Leasing Service* when "Company type" is *Leasing* is 66.7%. However, this conditional probability increases to 100%, if we only consider the input instances where "income" is in the range *[39, 41)* and "legal entity" is equal to *Private*. Hence, LAFF trains local models on subsets of $I^F(t)'$ to learn fine-grained dependencies.

More specifically, LAFF first selects a set of fields F^I that are independent from other fields in the probabilistic graph of M_0 (lines 4–8). For example, in block B of Figure 3.3, fields f_2 and f_3 are selected as they do not depend on other parent nodes (fields). We use the fields in F^I as the main fields to form partitions of $I^F(t)'$ having similar characteristics for two reasons. First, these fields are not intercorrelated since they do not directly and strongly depend on each other. Second, these fields are root nodes and influence the values of other fields in the BN; when the values on these fields are similar, we are likely to obtain a similar probability distribution for the other fields.

LAFF produces (lines 10–11) a set C of clusters of $I^F(t)'$ based on the fields in F^I. We assume that the clustering process reduces the data variation of $I^F(t)'$: models trained on these data, which show less statistical variation, may provide more accurate suggestions even when the size of each cluster is smaller than that of $I^F(t)'$. This process is called local modeling; it has been already applied in software engineering, e.g., to cluster software projects and mine project-specific relationships of software metrics [MBM+11].

To extract clusters from $I^F(t)'$, LAFF represents each historical input instance as a tuple of the form $\langle$values in F^I, input instance$\rangle$. It clusters these tuples based on the values in F^I using the k-modes algorithm. The optimal number of clusters k is automatically determined with the elbow method. LAFF runs k-modes within a range of k values (e.g., from 1 to 100) and determines the value of k that minimizes the average within-cluster distance with the cluster centroids (denoted with "cid" in block C of Figure 3.3). After clustering, LAFF trains a local BN model M_i (lines 12–15) based on the input instances in each cluster. These *local* models, denoted $M_1, \ldots, M_k$, capture specialized field dependencies on partitions of $I^F(t)'$.

The algorithm ends by producing a list $\mathcal{M}$ of BNs, where $\mathcal{M} = [M_0, M_1, \ldots M_k]$, and the set C of clusters of the historical input instances.

3.3.2.1 Application to the running example

Initially, LAFF trains a global model M_0 with the historical input instances in block A in Figure 3.3. Block B of Figure 3.3 shows an example of the learned field dependencies. Based on M_0, LAFF selects fields f_2:"income" and f_3:"legal entity" as the main fields for local modeling since they do not depend on other parent nodes (fields). LAFF clusters the historical input instances according to fields "income" and "legal entity" (block C of Figure 3.3). Three clusters are automatically identified with centroids "*[20, 22), Public*", "*[39, 41), Private*", and "*[39, 41), Public*" (k=3). We use circular, rectangular, and triangular icons to represent the historical input instances belonging to different clusters. LAFF trains three local models M_1, M_2, and M_3, based on these clusters; these three models are three distinct BNs capturing specialized field dependencies (as shown on the right of Figure 3.3). After the model building phase, LAFF outputs four models: a global BN model

Algorithm 2: Form Filling Suggestion

```
Input: Models $\mathcal{M} = [M_0, M_1, \ldots, M_k]$
       Clusters $C = \{I^F(t)'_1, I^F(t)'_2, \ldots, I^F(t)'_k\}$
       Filled fields $F^f = \{\langle f_1^f, v_1^f\rangle, \ldots, \langle f_m^f, v_m^f\rangle\}$
       Target field $f_p$
       Number of suggested values $n_r$
       Threshold $\theta$
Output: List of predicted values $V_p$ for $f_p$
1  $F^{f'} \leftarrow getPreprocessedData(F^f)$ ;
2  $D = \{d_1, \ldots, d_k\} \leftarrow calcClusterDistance(C, F^{f'})$;
3  Model $M_{cur} \leftarrow \mathcal{M}.get(M_0)$;
4  if $getNumOfMinDistance(D) = 1$ then
5      $i \leftarrow getMinDistanceID(D)$;
6      $M_{cur} \leftarrow \mathcal{M}.get(M_i)$;
7  end
8  List of Pairs $\langle v_p, pr\rangle$ of candidate values and probability distribution
    $Candidates = predictCandidates(M_{cur}, F^{f'}, f_p)$;
9  $Candidates^R \leftarrow getTopRanked(Candidates, n_r)$;
10 Bool $checkDep \leftarrow isMember(getParents(M_{cur}, f_p), F^f)$;
11 Bool $checkProb \leftarrow (getSumProb(Candidates^R) > \theta)$;
12 if $checkProb \lor checkDep$ then
13     foreach $v_{p_i}$ s. t. $\langle v_{p_i}, pr_i\rangle \in Candidates^R$ do
14         $V_p.append(v_{p_i})$;
15     end
16 end
17 return $V_p$;
```

M_0 and the three local BN models M_1, M_2, and M_3.

3.3.3 Form Filling Suggestion

The *form filling suggestion* phase occurs during the data entry session and assumes that the models in $\mathcal{M}$, built in the *model building* phase, are available. Given a target field f_p, LAFF selects a BN model $M \in \mathcal{M}$ and predicts possible values of f_p based on the already-filled fields F^f and their conditional dependencies captured in M. The main steps of the *form filling suggestion* phase are shown in Algorithm 2.

The algorithm takes as input a list of models $\mathcal{M}$, a set of clusters C, a set F^f of already-filled fields with their values, a target field f_p, and some auxiliary parameters representing the number of expected suggestions for f_p and an endorsing threshold. After pre-processing the filled fields in F^f using the techniques discussed in § 3.3.1 and obtaining the new set $F^{f'}$, LAFF computes the distance between the filled fields $F^{f'}$ and each cluster in C (line 2). The distance is defined as the dissimilarity measure adopted in the k-modes clustering algorithm used in the model building phase; it is the total number of mismatches between $F^{f'}$ and the centroid of each cluster on the corresponding fields.

LAFF attempts to select a local model from $M_1, \ldots M_k$, corresponding to the cluster with minimal distance, to predict the target field, since this model may capture the fine-grained characteristics of $F^{f'}$. In our example, this could be a model trained on the instances that have the same values for the fields "legal entity" and "income" in $F^{f'}$. However, a unique and optimal local model cannot always be found: given a partially filled data entry form, there could be cases for which the distance of the filled fields to different centroids is equal. For example, in Figure 3.3, local models M_2 and M_k are specialized for different values of the field "legal entity" (i.e., "Private" and "Public") but assume the same value for the field "income" ($[39, 41)$). Let us consider the case in which the set F^f contains only the field "income" (with a value equal to "40") and the target field is "primary activity". In this case, we cannot reliably select between the two models M_2 and M_k for prediction, since we have insufficient information to decide which model is "more local" (i.e., specialized) for this input (i.e., the distance to the two centroids is the same). One possible solution for this problem is ensemble learning, which considers the predictions of both models jointly (e.g., bagging) [Zho21]. However, this solution could significantly increase prediction time. Specifically, depending on the deployment configuration, in the worst case, the ensemble prediction time would be the sum of the prediction time of all k local models (i.e., all the local models are not specialized for the current input), which may exceed the acceptable response time for a practical application, as presented in section 3.4.3 and further discussed in section 3.5.2.1. Given the interactive nature of data-entry applications, having a short prediction time is important. Hence, when LAFF finds more than one minimal distance and no single cluster is particularly suited for the current input, it selects M_0 for prediction, since it is trained with the entire set of historical input instances (lines 3–7).

After selecting the most appropriate model for prediction, LAFF predicts the candidate values for the target field (line 8) and ranks the topmost n_r values, according to their probability distribution (line 9).

3.3.3.1 Endorsing

During the data entry session, the filled fields in the current input instance do not always provide enough information to predict values for the target field, leading to inaccurate suggestions. Such a situation can occur because of two reasons. One reason is that the filled fields may not have explicit dependencies with the target field, according to the probabilistic graph. For example, in Figure 3.3, f_2 is independent from f_3; LAFF will not accurately infer f_3 merely with the knowledge of f_2. Another reason is that there may not be enough historical input instances to learn the conditional probability between two fields for specific values. For example, in the example in Figure 3.3, we have no historical input instance with values of field "income" greater than 41; such value provides limited information to infer other fields.

In the context of automated form filling, users might be reluctant to use an automated form filling tool, if the tool provides many inaccurate suggestions which users can hardly find the correct value they intend to fill in. To avoid such a situation, LAFF includes a *heuristic-based endorser*, which decides whether the suggestions determined in the previous step are accurate enough to be returned to the user.

To deal with the first cause of inaccurate suggestions, LAFF analyzes the dependency between the filled fields and the target field (line 10), to check whether the target field directly depends on one of the filled fields in the BN. More precisely, function *getParent* computes a list of parent fields the target field directly depends on; function *isMember* checks whether any of the filled fields is in the parent field list. The result of this check is saved in the Boolean flag $checkDep$, which is true when the target field directly depends on one of the filled fields. A direct dependency indicates that the filled fields can reliably determine the value of the target field.

To deal with the second cause of inaccurate suggestions, LAFF analyzes the predicted probability distribution of the values for the target field. For a probability model like BN, the probability of each value is inferred based on the information from the filled fields. LAFF computes the sum of the top-n_r probability values in the distribution through function *getSumProb*. If this value is larger than a user-defined threshold θ, it means LAFF may have enough information for variable inference; the result of this check is saved in the Boolean flag $checkProb$ (line 11). From a practical standpoint, threshold θ reflects how much uncertainty users are willing to accept regarding the suggestions provided by LAFF.

If one of the flags $checkProb$ and $checkDep$ evaluates to *true*, LAFF populates the list of suggested values to be returned to the user based on the top-

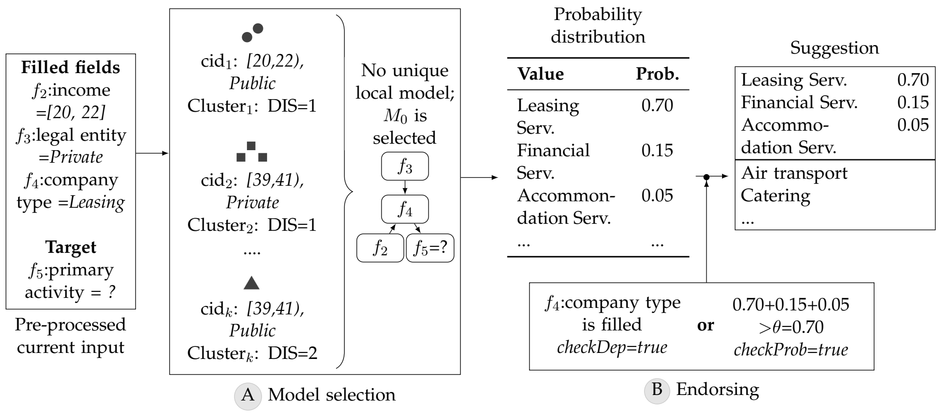


Figure 3.4: Workflow for Form Filling Suggestion

ranked candidate values; otherwise, LAFF returns an empty list (lines 12–15). For example, assuming $\theta = 0.70$, $n_r = 3$, and the probability for the top-3 candidate values as shown in the top right corner of Figure 3.2, the sum of the top-3 probability values (returned by $getSumProb$) is $0.70+0.15+0.05 = 0.90$; $checkProb$ corresponds to the evaluation of $0.90 > 0.70$, which is *true*; hence, LAFF decides to yield the list of suggestions to the user.

3.3.3.2 Application to the running example

Given the new input instance shown on the left side of Figure 3.4 (i.e., the instance "income"=*[20, 22)*, "legal entity"=*Private*, and "company type"=*Leasing*, as obtained after pre-processing), LAFF suggests the possible values of "primary activity". As shown in block A of Figure 3.4, LAFF first attempts to select a unique local model by calculating the distance between the current input instance and the centroids of the three clusters generated in block C of Figure 3.3; however, such a local model cannot be found because the distances with "cid$_1$" and "cid$_2$" are both 1. Hence, LAFF uses M_0 for prediction. According to the variable inference method in BN (explained in section 2.1), LAFF outputs the probability distribution of the candidate values for the field "primary activity". Let us assume, as an example, that the probability distribution is "*Leasing Service*=0.70, *Financial Service*=0.15, *Accommodation Service*=0.05, ...". By means of the endorser module (block B of Figure 3.4), LAFF uses this probability distribution to decide whether to present the suggestions to the user. For example, let us further assume the data quality engineers in the bank set θ to 0.70 and configures LAFF to suggest three values. On the one hand, LAFF finds that the target field f_5:"primary activity" directly depends on f_4:"company type", which was already filled by the user; the *checkDep* flag is *true*. On the other hand, the sum of the top-3 probability values is 0.90, which is higher than θ; the *checkProb* flag is *true*. Since the endorser module endorses a suggestion when one of these two flags evaluates to true, LAFF provides a suggestion to the user: the three values above are put to the top of the list while the other candidate values are presented in their original order (e.g., alphabetically).

3.4 Evaluation

In this section, we report on the evaluation of our approach (LAFF) for automated form filling. First, we assess the overall accuracy of LAFF in suggesting appropriate values to automatically fill in the fields of data entry forms, and compare it with state-of-the-art form filling algorithms. We also assess the performance of LAFF, in terms of training time and prediction time, for practical applications. Then, we evaluate how the use of local modeling (in the *model building* phase) and heuristic-based endorser (in the *form filling suggestion* phase) affect the accuracy of LAFF. Last, we assess the impact of the number of filled fields and the size of the training set on the effectiveness of LAFF.

More specifically, we evaluated LAFF by answering the following research questions:

RQ1 *Can LAFF provide accurate suggestions for automated form filling, and how does it compare with state-of-the-art algorithms?*

RQ2 *Is the performance of LAFF (in terms of training time and prediction time) suitable for practical application in data-entry scenarios?*

RQ3 *What is the impact of using local modeling and heuristic-based endorser on the effectiveness of LAFF?*

RQ4 *What is the impact of the number of filled fields on the effectiveness of LAFF?*

RQ5 *What is the impact of the size of the training set on the effectiveness of LAFF?*

3.4.1 Dataset and Settings

3.4.1.1 Datasets

We evaluated LAFF using a public dataset in the biomedical domain (dubbed NCBI) and a proprietary dataset, extracted from a production-grade enterprise information system, provided by our industrial partner (dubbed PROP).

Table 3.1: Information about the Fields in the Datasets

Dataset	# of fields	# of instances	Name of categorical fields (# of candidate values)	Value frequency top-1	top-5%
NCBI	26	74105	sex(3), tissue(68), cell-line(50), cell-type(63), disease(84), ethnicity(40)	40.8%	59.4%
PROP	33	174446	title(18), sex(3), legal capacity(7), country(208), first nationality(206), civil status(8), matrimonial regime(6), activity(13), status(15), function(41), contract(8), field of activity(75), primary activity(3), country of activity(198)	48.4%	65.6%

The NCBI dataset contains the metadata for diverse types of biological samples from multiple species [BCG+12]. We selected this dataset because it has been used in a previous study on metadata suggestion for biomedical datasets [MROE+19], which provided also the design of the corresponding data entry form in the CEDAR workbench [GOMR+17]. More specifically, following the evaluation methodology described in [MROE+19], we considered the subset of the NCBI dataset related to the species “Homo sapiens” and the corresponding data entry form based on the specification of the BioSample Human package v1.0[2]. We downloaded the dataset from the official NCBI website[3]. In the dataset, the data is organized as a table. Each row is an input instance filled by a user. Retrieving the mapping between column names and field names was trivial since the column names in the dataset are the same as the field names. As shown in Table 3.1, the NCBI dataset has 26 fields, six of which are categorical. These categorical fields have between 3 and 84 candidate values to be selected by users. We calculated the frequency by which users select different values during form filling: the most frequent (i.e., top-1) and the top-5% most frequent values are selected, on average, respectively in 40.8% and 59.4% of the instances for different categorical fields. Given the sparseness of the dataset (caused by the optional fields), as suggested in [MROE+19], we identified the empty values (e.g., “n/a”, “null”), and only retained records with at least three fields (out of six) with non-empty values; in total, the NCBI dataset contains 74 105 input instances.

The PROP dataset contains customer data that are provided through a web-based data entry form, which is filled out upon creation of a new customer account. We extracted the dataset from the distributed database of

[2] https://submit.ncbi.nlm.nih.gov/biosample/template/?package=Human.1.0&action=definition

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

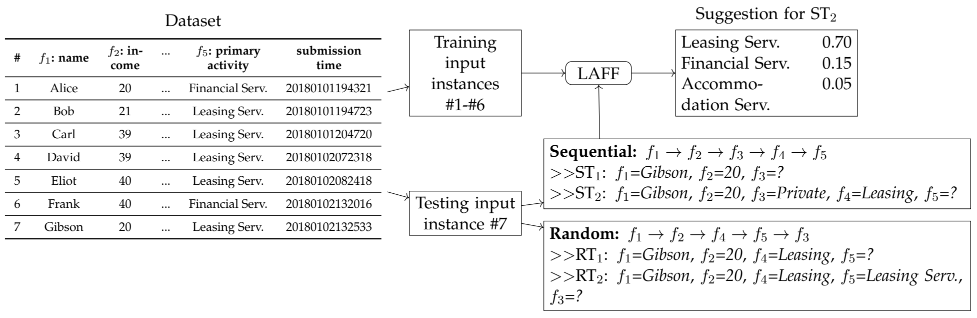
our industrial partner, where all the input instances of a certain form are organized as a database table. Each row in the table is an input instance and each column represent a form field. We identified the mapping between the column names in the table and the field names in the data entry form by consulting the available software documentation. As shown in Table 3.1, the PROP dataset has 33 fields, 14 of which are categorical (with the number of candidates values ranging from 3 to 206). In terms of frequency according to which users select different candidate values, the top-1 and the top-5% most frequent values are selected, on average, respectively in 48.4% and 65.6% of the instances. According to the form design, eight of the categorical fields are mandatory to be filled; hence, we do not remove spare records as done for the other dataset; in total, the PROP dataset contains 174 446 input instances.

We remark that both datasets represent the input instances from real-world data entry forms (i.e., the NCBI platform and a production-grade enterprise information system). The number of fields in these systems is comparable with or larger than the data entry forms used in the related work. For example, we calculated the average number of fields of data entry forms in the TEL-8 dataset, a manually collected dataset with 447 web forms (with no input instances), which is used in the literature on form filling [AGLH10, Jou19]. In this dataset, each form has 6.39 fields on average. The data entry forms in our study are more complex, ranging from 26 to 33 fields of different types.

3.4.1.2 Dataset Preparation

For the two datasets, as discussed in section 3.2.3, all the categorical fields are the targets for automated form filling. However, we excluded the fields with less than 10 candidate values (e.g., “sex”, which has only three values in both datasets) as users may easily browse all the values in these fields, without the need for form-filling automation. The threshold for excluding fields was determined together with the data quality engineers and some data entry operators of our partner. We find the majority of categorical fields we evaluated are related to certain domains or business processes; they include fields “tissue”, “cell-line”, “cell-type”, “disease” and “ethnicity” for the biological domain, and fields “activity”, “status”, “function”, “field of activity”, and “country of activity” for the financial domain. These fields are more difficult to fill than basic user information (e.g., name, sex, and age), since users need to understand the meaning of candidate values.

Since both datasets automatically recorded the submission time of each input instance. we split the dataset into two subsets containing 80% and



#	f_1: name	f_2: income	...	f_5: primary activity	submission time
1	Alice	20	...	Financial Serv.	20180101194321
2	Bob	21	...	Leasing Serv.	20180101194723
3	Carl	39	...	Leasing Serv.	20180101204720
4	David	39	...	Leasing Serv.	20180102072318
5	Eliot	40	...	Leasing Serv.	20180102082418
6	Frank	40	...	Financial Serv.	20180102132016
7	Gibson	20	...	Leasing Serv.	20180102132533

Leasing Serv.	0.70
Financial Serv.	0.15
Accommodation Serv.	0.05

Sequential: $f_1 \rightarrow f_2 \rightarrow f_3 \rightarrow f_4 \rightarrow f_5$
>>ST$_1$: f_1=*Gibson*, f_2=*20*, f_3=*?*
>>ST$_2$: f_1=*Gibson*, f_2=*20*, f_3=*Private*, f_4=*Leasing*, f_5=*?*

Random: $f_1 \rightarrow f_2 \rightarrow f_4 \rightarrow f_5 \rightarrow f_3$
>>RT$_1$: f_1=*Gibson*, f_2=*20*, f_4=*Leasing*, f_5=*?*
>>RT$_2$: f_1=*Gibson*, f_2=*20*, f_4=*Leasing*, f_5=*Leasing Serv.*, f_3=*?*

Figure 3.5: Example of Dataset Preparation (Training and Testing Sets)

20% of input instances based on their submission time, used respectively for training and testing. The input instances (excluding the information of the submission time) in the training set are used to train LAFF. As for the testing input instances, since there is no information on the actual filling order used to input the data, we considered two form filling orders to simulate the data entry session. More specifically, we simulated two types of filling scenarios: "sequential filling" and "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 [FS04]. It simulates the logical order many users follow to fill out forms, especially when they use a keyboard to navigate form fields [Mic13]. The latter represents the scenario when users may select any field as the next target, and even go back to modify already filled fields. These two form filling orders represent two opposite extremes[4] of user behavior during a real data-entry session. We simulated random filling by randomly generating an order for each testing input instance. In both form filling scenarios, the filled fields considered by LAFF are the fields that precede each target. For each target field, we consider the actual value filled by the user as the ground truth.

[4]For some large data entry forms, UI designers can semantically partition related fields into sections. Users can then move between sections in sequential order, while using the random order to fill fields within a section. This is a "middle-ground filling" order, which sits between "sequential filling" and "random filling". We have not evaluated this scenario since it requires additional knowledge about the partitioned sections, which was not available for the two datasets we have considered.

3.4.1.3 Dataset Preparation - Example of Application

Figure 3.5 shows an example of application of our dataset preparation method for the training and testing sets. Let us consider a dataset containing seven input instances submitted through a data entry form, shown on the left-hand side of Figure 3.5. Following the running example introduced in section 3.2.3, the form has five fields, two of which are categorical (e.g., f_3: “legal entity” and f_5:“primary activity”). We split the dataset into the training and testing sets according to the submission time: we take 80% of input instances (i.e., #1-#6) to train LAFF; the testing set contains the remaining 20% of the instances, in this case input instance #7. The testing set is further processed to simulate the two types of filling scenarios. When using the sequential filling order, users fill the data entry form following the `tabindex` of fields in the form (e.g., from f_1 to f_5 sequentially): starting from the input instance #7, we generate test instances ST_1 and ST_2 for categorical fields f_3 and f_5, respectively. For each categorical field (i.e., the target), the actual value filled by the user is the ground truth (e.g., the ground truth for the field f_5 is ‘Leasing Serv.’). When using the random filling order, we randomly generate a field order for each input instance (e.g., $f_1 \to f_2 \to f_4 \to f_5 \to f_3$ for the input instance #7); based on this order, we then generate test instances RT_1 and RT_2.

3.4.1.4 Implementation and Settings

We implemented LAFF as a Python program; we used the open-source library `pgmpy` [AP15] for working with Bayesian networks.

We configured LAFF (through parameter n_r in Algorithm 2) to suggest the top 5%, most likely values for each target field. Based on the number of candidate values for each field in the datasets (indicated in parentheses in the rightmost column of Table 3.1), suggesting the top 5% values means showing between one (for field “activity” in the PROP dataset) and ten (for field “country” in the PROP dataset) suggested values to users. This is in accordance with other recommender systems in software engineering, in which only a list of few candidates is suggested for consideration [YLX$^+$16, PCJ$^+$17]. We set the threshold θ to 0.7 based on the feedback received by data entry operators and data quality engineers of our partner.

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 PROP dataset[5], we performed them on a

[5]Due to the data protection policy of our partner, we were obliged to run the experiments on the PROP dataset using an on-premise, dedicated server that, however, could not be used

server running CentOS 7.8 on a 2.60 GHz Intel Xeon E5-2690 processor with 125 GB memory.

3.4.2 Effectiveness (RQ1)

To answer RQ1, we assessed the effectiveness of LAFF to suggest appropriate values for each of the target fields in the dataset. We compared LAFF with MFM (most frequent model), ARM (association rule mining) [MROE+19], NaïveDT (naïve application of decision trees), and FLS (first letter search), which are able to provide suggestions under different form filling orders:

1. MFM is a widely-used form filling algorithm, which suggests possible values of a target field based on their frequency in historical input instances.

2. ARM is a state-of-the-art algorithm for form filling. ARM uses historical input instances to mine association rules with a minimal level of support and confidence; it matches the filled fields with mined association rules, and suggests the consequents of the matched rules to users.

3. NaïveDT is a naïve application of decision trees for form filling. We use decision trees because this type of model has been already used in the form filling literature [HS94]. Given a target field, this approach takes a subset of the remaining fields as filled fields (i.e., features); it then trains a decision tree for each feature-target combination. During form filling, based on the filled fields and the target field, NaïveDT selects the decision tree trained on the same feature-target combination in order to predict the target field. We considered this naïve application of decision trees because previous work [HS94] has shown that the effectiveness of a single decision tree trained for each target field is poor (see also section 3.6); our preliminary evaluation has also confirmed this.

4. FLS simulates form filling in categorical fields through a "typing" function. This function allows users to type the first letter of the candidate value they intend to fill (i.e., the first letter of the ground truth in this study). FLS filters the list of candidate values based on this letter and presents the refined candidate values as suggestions.

We did not consider other approaches for automated form filling, since they rely on additional information beyond the input values provided in the

to store external data (like the NCBI dataset).

past for the same form. For example, they reuse the values filled in other software systems [AGLH10], extract information from text files (e.g., a CV file to fill job search sites) [TCdSdM10], or refactor forms for effective form filling [CCC+11]. An empirical comparison with these techniques is not feasible, since such additional knowledge is not always available during form filling; moreover, LAFF does not assume the existence of such knowledge. We discuss the differences between LAFF and these related approaches in section 3.6.

3.4.2.1 Choosing effectiveness metrics

We reviewed the main metrics used for evaluating the effectiveness of building a suggestion list. More specifically, we investigated the metrics used in the recommender systems area because, similar to automated form filling, many software applications use recommender systems to support software stakeholders in their decision-making while interacting with large information spaces [RWZ09] (e.g., locating faulty code snippets in software projects [YBL14]). Table 3.2 shows, for each metric we reviewed, its dimension, description, and rationale.

Metrics for evaluating recommender systems span over four dimensions, including diversity [KP17], novelty [CVW11, KB16], accuracy [HKTR04], and coverage [GDBJ10, HKTR04]. As shown in Table 3.2, diversity and novelty focus on the dissimilarity among the suggested items: the former by looking at pairwise dissimilarity, and the latter by determining the difference between the currently and previously suggested items. Both metrics can be applied in contexts where more than one relevant item can be suggested. As for assessing accuracy, precision and recall are the most common metrics [HKTR04]; they measure the ratio of correctly suggested items. However, precision and recall ignore the exact ranking of items as only the correct or incorrect classification is measured [STL11]. Other common accuracy metrics include MRR and MAP [STL11, KJJ18, HKTR04], which are designed to evaluate a list of suggested items. MRR calculates the rank of the first relevant item, and MAP measures the average precision of relevant items at different positions. Regarding coverage, two definitions have been proposed in the literature [GDBJ10]: catalog coverage and prediction coverage. Catalog coverage measures the length of a list of suggested items relative to its maximum length; prediction coverage calculates the ratio of targets for which an algorithm provides suggestions over the total number of targets requiring suggestions. Finally, in the literature, some metrics are also proposed to combine different metrics from the same dimension to evaluate the trade-off between

Table 3.2: Main metrics used in the recommender system area for evaluating the effectiveness of building a list of suggestions

Dimension	Description	Intuition
Diversity	Diversity metrics generally measure the average dissimilarity between all pairs of items in the suggestion list [KP17].	The suggested items can cover a broad area of the information space to increase the chance of satisfying the user's information need. For example, with a movie recommender system, users may hope to get relevant items from different genres (e.g., "comedy", "romance")
Novelty	The novelty of an item is typically estimated by the inverse of its popularity (e.g., measured by the number of ratings it has received); novelty metrics measure the ratio of the suggested relevant items that have low popularity [KB16].	A tool has the ability to suggest relevant items that are unknown to users. For example, a movie recommender system should have the ability to recommend "new" movies that users did not watch or know before.
Accuracy	*Precision* measures the fraction of suggested relevant items among all the suggested items [HKTR04].	The suggestion list only contains relevant items.
	Recall measures the fraction of suggested relevant items among all the relevant items [HKTR04].	The suggestion list contains all the relevant items regarding a target.
	MAP (Mean Average Precision) measures the mean of the average precision at the rank of each relevant item [HKTR04].	All the relevant items can be ranked at the top of a suggestion list.
	MRR (Mean Reciprocal Rank) measures the mean of the reciprocal rank at which the first relevant item was suggested in a list [MROE$^+$19].	The relevant item can be ranked at the top of a suggestion list.
Coverage	*Catalog coverage* measures the ratio of the items suggested to users over the total number of candidate items of a given target [GDBJ10].	A tool can avoid suggesting a long list of items to users (e.g., only the top 5% suggested items are presented to the users)
	Prediction coverage measures the ratio of suggestions provided by a tool over the total number of targets requiring suggestions [GDBJ10].	A tool can avoid making "useless" suggestions to users. Low-confidence suggestions with many unrelated items should be filtered out to fit the user's interests.
Combined	*F_1-score* is the harmonic mean of the precision and recall [APGG14].	A tool can suggest (recall) and only suggest (precision) all the relevant items to users.
	Quality of suggested items is the product between their similarity to the user's query and the diversity of the items [APGG14].	The items suggested in the suggestion list are relevant to the user's query (similarity) and at the same time different from each other (diversity).

them. For example, metrics like F_1-score and the quality metric [APGG14] have been proposed to balance the weight of precision and recall (accuracy dimension) or similarity and diversity (diversity dimension), respectively.

According to a previous study [RWZ09], an effective recommender system in software engineering (e.g., a form filling system) is expected to avoid "helpless" suggestions that may be ignored by users, but provide users a large number of "helpful" suggestions. Considering the dimensions in Table 3.2, metrics for diversity and novelty are not applicable for form filling, as most categorical fields only contain a single correct value for a user to select. The accuracy metrics can be used to assess the "helpfulness" of suggestions. Since LAFF suggests a list of values for users to select the correct one, MRR is the most appropriate metric in our context, since it evaluates if the correct value is ranked at the top of a suggestion list. Regarding coverage, we select the prediction coverage to evaluate the extent to which the endorser module of LAFF can avoid "helpless" suggestions; it calculates the frequency of suggestions made by LAFF when required to make one. Since MRR and prediction coverage belong to different dimensions, we separately evaluate the two metrics instead of combining their results with a single score.

In the following, we provide the definition of MRR and prediction coverage.

MRR (Mean Reciprocal Rank) is defined as:

$$MRR = \frac{1}{|S|}\sum_{i=1}^{S}\frac{1}{k_i}, \tag{3.1}$$

where $|S|$ is the number of target fields that the algorithm provides suggestions, and k_i is the position of the first correct value in the i-th suggestion.

Prediction coverage rate is defined as:

$$PCR = \frac{|S|}{|S_{all}|}, \tag{3.2}$$

where $|S|$ is again the number of target fields that the algorithm provides suggestions, and $|S_{all}|$ is the total number of target fields in all the testing input instances [HKTR04, GDBJ10].

3.4.2.2 Methodology

To assess the effectiveness of the various form filling algorithms, we computed MRR and the prediction coverage rate PCR.

We remind the reader that LAFF uses the filled fields (i.e., features) of each test instance to predict the value of a target field. In our datasets, all the target categorical fields only have a single correct value (e.g., in Figure 3.5, *Leasing Serv.* is the ground truth for the field f_5 of the input instance #7).

For each test instance, we checked the position of the correctly suggested value (i.e., the value that corresponds to the ground truth) in each suggestion list and computed the reciprocal rank of the correct value. If no correct value was found, we set the reciprocal rank to zero. For example, in Figure 3.5, LAFF suggested three values; the reciprocal rank of the suggestion for ST_2 is 1 since the user can find the correct value to fill in first position (i.e., $k_i = 1$). The MRR value was computed as the mean of the reciprocal ranks for all the suggestion lists. Given a test set, we calculated the average MRR value for different targets. Concerning PCR, we counted the number of target fields for which an algorithm provided suggestions (i.e., a list of suggested values) and the number of target fields for which no suggestion was provided. PCR was computed as the percentage of target fields receiving suggestions over the total number of target fields.

As there is no publicly available implementation of the baselines (ARM, MFM, NaïveDT, and FLS) for form filling, we implemented them from scratch.

In the case of ARM, we set the minimum acceptable support and confidence to 5 and 0.3, respectively, consistent with previous work [MROE+19].

We implemented NaïveDT using the open-source library `scikit-learn` [PVG+11a]. For the NCBI dataset, after the preprocessing step (see section 3.3.1) we obtained six fields; all of them are categorical. According to the discussion in section 3.3.2, we have $n = 6$ and $t = 6$, resulting in $6*(2^{6-1}-1) = 186$ feature-target combinations (i.e., decision trees to train). For the PROP dataset, after the preprocessing step, we obtained 15 fields, among which 14 are categorical (i.e., $n = 15$ and $t = 14$). This leads to $14 * (2^{15-1} - 1) = 229\,362$ decision trees to train.

Regarding FLS, we ranked the candidate values for a given target field alphabetically and refined these values by the first letter the user intends to fill. We assume that the first letter of the value of the ground truth is what the user intends to fill. Based on this assumption, FLS suggests the candidate values that start with the same letter as the ground truth.

We set a timeout of 24 hours to train each algorithm. This timeout value reflects the realistic situation in which the algorithm gets daily updates of its models, empowered with the information derived from new input instances collected throughout the day.

3.4.2.3 Results

Table 3.3 shows the effectiveness of the various algorithms for the two form filling scenarios. We remark that NaïveDT timed out during the training

Table 3.3: MRR and PCR of Form Filling Algorithms (t/o: timeout; N/A: not applicable) .

	Alg.	Sequential		Random		Train	Predict (ms)	
		MRR	PCR	MRR	PCR	(s)	avg	min–max
NCBI	MFM	0.42	1.00	0.42	1.00	0.03	0	0
	ARM	0.47	1.00	0.53	0.99	1.26	120	8–409
	NaïveDT	0.52	1.00	0.53	1.00	23	1	1–1
	FLS	0.55	1.00	0.54	1.00	N/A	0	0
	LAFF	0.74	0.70	0.73	0.70	546	15	5–44
PROP	MFM	0.64	1.00	0.64	1.00	0.15	0	0
	ARM	0.67	0.95	0.65	0.96	15	878	6–3074
	NaïveDT	t/o	t/o	t/o	t/o	t/o	t/o	t/o
	FLS	0.53	1.00	0.58	1.00	N/A	0	0
	LAFF	0.78	0.86	0.81	0.87	3652	138	9–317

phase on the PROP dataset[6]

In terms of coverage rate (columns PCR), MFM, ARM, NaïveDT, and FLS provide suggestions on almost all the target fields ($PCR \approx 1$), while LAFF achieves a PCR value ranging from 0.70 to 0.87 on the two datasets. The lower PCR value of LAFF is ascribable to its endorsing module, which discards low-confidence suggestions.

As to the accuracy of these suggestions, MFM, ARM, and NaïveDT achieve MRR values ranging from 0.42 to 0.53 on the NCBI dataset; MFM and ARM achieve MRR values ranging from 0.64 to 0.67 on the PROP dataset. LAFF substantially outperforms MFM, ARM, and NaïveDT, achieving a MRR value above 0.73 in both datasets. The improvement in MRR value obtained by LAFF over NaïveDT is $+22$ pp on the NCBI dataset; the one over ARM is $+11$ pp on the PROP dataset for the sequential filling scenario. For the random filling scenario, the improvement over NaïveDT and ARM is $+20$ pp on the NCBI dataset; the improvement over ARM is $+16$ pp on the PROP dataset. According to Table 3.3, LAFF also outperforms the interaction-based approach FLS by $+19$ pp to $+25$ pp on the two datasets for different filling scenarios. The reason is that, after refining the candidate values by typing letters, users could still have dozens of candidate values with the same initial letter to check. For example, when users type "L" to find the country "Luxembourg", the form returns 9 results (e.g., "Laos", "Latvia", "Lebanon")

[6]For completeness, based on our preliminary evaluation, we note that using a single decision tree trained for each target field (as done in previous work [HS94]) would have avoided the time-out on the PROP dataset but would have yielded worse effectiveness results than the lowest values reported in Table 3.3, confirming the trend already reported by Hermens and Shlimmer [HS94] (see also section 3.6).

where “Luxembourg” is the last one in the refined list; in contrast, LAFF can rank “Luxembourg” as the first country for users to choose, leading to much higher MRR values. Compared with FLS, LAFF has two advantages. First, many users lack a detailed conceptual model of the software system [HKOP03] defined by the requirements analysts and domain experts. They may not remember all the candidate values predefined in a categorical field, leading to a potential lengthy search process [HKOP03]. In this case, LAFF can directly provide the most-likely suggestions for users to choose based on the filled fields. Second, LAFF is compatible with FLS. Based on the highly accurate suggestions made by LAFF, users can continue refining the suggested list with FLS when needed to further accelerate their form filling process. For example, when users type “L” after LAFF’s suggestion, only country names starting with “L” can be retained (e.g., “Luxembourg”, “Laos”, “Latvia”, “Lebanon”).

We use the Mann-Whitney U test to assess the statistical significance of the difference between the MRR values of LAFF and the baselines, with a level of significance $\alpha = 0.05$. The results show that LAFF always achieves a statistically higher MRR value than the baselines for the two form filling scenarios on both the NCBI and PROP datasets (p-value < 0.01).

These results have to be interpreted with the usage scenarios of a recommender system. Previous studies show that, for a recommender system, inaccurate suggestions increase users’ decision time and the risk of making wrong decisions [OEDK18]. The MRR and PCR values achieved by LAFF show that the suggestions provided by LAFF allow users to find the correct value among the top-ranked suggested values.

3.4.2.4 Error analysis

We further analyzed the suggestions made by LAFF, to identify the cases in which it does not perform well. We recall that in our experiments, LAFF suggested the top 5% most likely values for each target field. On the NCBI dataset, LAFF captures the correct value in the top 5% suggested values for 79.0% of the suggestions when using sequential filling and for 79.9% of the suggestions when using random filling. On the PROP dataset, the correct value is in the top 5% suggested values for 89.5% (sequential filling) and 90.1% (random filling) of the suggestions. Overall, 79.0% to 90.1% of the suggestions made by LAFF allow users to find the correct value among the 5% top-ranked suggested values. For the remaining incorrect suggestions (around 21% on the NCBI dataset and 9.9% on the PROP dataset), in which the correct value is not in the top 5% suggested values, we identified the following main reasons.

First, LAFF tends to provide incorrect suggestions when the number of filled fields used for prediction is small. Specifically, on the NCBI dataset, 33.1% of the incorrect suggestions when using sequential filling and 53.1% of the incorrect suggestions when using random filling were made when there was only one filled field; on the PROP dataset, the ratio is 17.4% and 23.5%, respectively. With few filled fields, LAFF may not get enough knowledge (i.e., the information of dependent field values) for prediction. This affects both the variable inference step within BNs and the behavior of the endorser module. In this case, LAFF tends to use the most frequent value in the target field for prediction, since this value has a higher prior probability. One possible way to mitigate this issue, to be investigated as part of future work, could be to define form refactoring techniques that allow users to first fill fields that provide additional knowledge used to predict the values of other fields. We analyze the impact of the number of filled fields on the effectiveness of LAFF as part of RQ4 (§ 3.4.5).

Second, incorrect suggestions are caused by the number of training input instances. Due to optional fields, users may not provide values for all the fields. For example, for the field "ethnicity" in the NCBI dataset, only 15.6% of input instances contain a non-empty value. The sparseness of the filled values for a field leads to a small number of training input instances to learn the corresponding dependency. Continuing the example, the MRR values for the field "ethnicity" are 0.608 and 0.553 for the sequential filling and random filling scenarios, respectively, thus leading to many incorrect suggestions. To mitigate this problem, as part of future work, instead of using a single threshold for endorsing suggestions, we could modify the endorser used in LAFF to support field-specific thresholds. We analyze the impact of the size of the training set on the effectiveness of LAFF as part of RQ5 (§ 3.4.6).

Third, the number of options (i.e., candidate values) for a field *may* affect the effectiveness of LAFF. To investigate this, we computed the correlation between the number of options and the MRR value for a field, considering the MRR values achieved in the random filling scenario[7]. The resulting Pearson correlation coefficient is -0.09 on the NCBI dataset (p-value=0.722) and -0.477 on the PROP dataset (p-value=0.001), thus showing no correlation for NCBI and a moderate but significant correlation for PROP [Ema99]. The dif-

[7]We did not consider the sequential filling scenario, since it could introduce some bias in our analysis: The number of filled fields to predict for each target field is different, as it depends on the tabindex order of the field.

ference in results between datasets can be easily explained by the fact that the variance in number of options is very low for NCBI (228.8), while it is much larger for PROP 6713.25. These results therefore suggest that, when a field has more options, LAFF tends to provide more incorrect suggestions.

To conclude, *the answer to RQ1 is that LAFF can yield a large number (with a PCR value ranging from 0.70 to 0.87) of accurate suggestions, with a MRR value above 0.73, significantly outperforming state-of-the-art approaches.*

3.4.3 Performance (RQ2)

To answer RQ2, we measured the execution time required to perform the model building phase of LAFF (i.e., training time), as well as the time to predict a target field (i.e., prediction time). The training time indicates the feasibility of using LAFF in contexts where the training set (i.e., the set of historical input instances) is updated often as new input instances are recorded in the system. The prediction time indicates how fast LAFF can provide form filling suggestions during a data entry session.

3.4.3.1 Methodology

We used the same settings (i.e., form filling scenarios) as in RQ1. We computed the training time as the time to build all BN models over the historical input instances. The prediction time is the average time (over the various target fields) taken to provide a suggestion for one input instance using locally deployed models. We also compared LAFF with the MFM, ARM, and NaïveDT algorithms. Notice that FLS does not require any training and the prediction can be considered instantaneous.

3.4.3.2 Results

The results are shown in the last two columns in Table 3.3, in term of training time (column *Train*) and the prediction time (column *Predict*, with subcolumns indicating the average, minimum, and maximum values, when applicable) for the two datasets.

The training time of LAFF is much higher than the one of MFM, ARM, FLS (and NaïveDT on the NCBI dataset); LAFF takes 546 s and 3652 s to train models on the NCBI and PROP datasets, respectively. This can be easily explained since LAFF trains several models, as explained in section 3.3.2. Although NaïveDT took only 23 s to train all the decision trees on the NCBI dataset, it timed out on the PROP dataset. As for prediction time, that of LAFF is higher than that of MFM, NaïveDT, and FLS. MFM and FLS directly suggest the frequency-based (for MFM) or matching-based (for FLS) value list to users. For NaïveDT, the prediction time includes the time to select the appropriate model (based on the feature-target combination) among the trained models. However, the prediction time of LAFF is, on average, faster than the one of ARM by 105 ms (15 ms vs 120 ms) over the NCBI dataset and by 740 ms (138 ms vs 878 ms), over the PROP dataset. The Mann-Whitney U test also confirms that the differences in prediction time between LAFF and the baselines are statistically significant (p-value < 0.01 for the two datasets).

These results have to be interpreted taking into account the usage scenarios of our approach. The training time has to be considered when training the models with new data, i.e., the new input instances recorded in the system since the last execution of the model building phase. This task is performed *offline* and *periodically* (e.g., once a day), so a training time of the order of one hour is acceptable from a practical standpoint.

Given the interactive nature of data-entry applications, having a short prediction time is much more important for an automated form filling approach. According to human-computer interaction principles [Hee00], users feel a system reacts instantaneously when its response time is within 100 ms and feel they are seamlessly interacting with the software system when the response time is within 1 s. In our context, the prediction time of LAFF depends on the computational power at our disposal and the complexity of the trained BN models (i.e., the number of nodes and the size of probability tables). The experiments using the proprietary dataset (extracted from a production-grade system) show that LAFF is fast enough (requiring at most 317 ms) to provide real-time suggestions during data entry sessions.

The answer to RQ2 is that the performance of LAFF, with a training time of about one hour (or less) and a prediction time of at most 317 ms, *is suitable for practical application in data-entry scenarios.*

3.4.4 Impact of Local Modeling and Endorser (RQ3)

LAFF has two important modules: the local modeling module, which builds local models based on local field dependencies of partitions of historical input instances (section 3.3.2); the endorsing module, which uses heuristic rules to remove possibly inaccurate suggestions (section 3.3.3). To answer RQ3, we assessed the impact of these two modules on the effectiveness of LAFF.

Table 3.4: Effectiveness of LAFF with Different Modules

ID	Module		NCBI				PROP			
			Sequential		Random		Sequential		Random	
	L	E	MRR	PCR	MRR	PCR	MRR	PCR	MRR	PCR
LAFF-LE	✗	✗	0.54	1.00	0.56	1.00	0.74	1.00	0.77	1.00
LAFF-E	✓	✗	0.60	1.00	0.60	1.00	0.76	1.00	0.79	1.00
LAFF-L	✗	✓	0.55	0.83	0.61	0.77	0.78	0.86	0.79	0.87
LAFF	✓	✓	0.74	0.70	0.73	0.70	0.78	0.86	0.81	0.87

3.4.4.1 Methodology

As shown in Table 3.4, in the two sub-columns of column *Module*, we considered four variants of LAFF, to reflect possible configurations with the two modules. In the table, *L* refers to the local modeling module and *E* refers to the endorsing module; symbols '✓' and '✗' indicate whether the variant of LAFF includes or not a certain module, respectively. When the local modeling module is disabled (denoted by LAFF-L), it means LAFF only uses the global model for prediction; when the endorsing module is disabled (denoted by LAFF-E), it means we do not scrutinize (and possibly discard) the suggestions provided by LAFF. If both modules are disabled (denoted by LAFF-LE), LAFF becomes a plain BN model trained on the entire set of historical input instances. We ran the vanilla version of LAFF (i.e., the one presented in section 3.3) and the additional variants using the same settings as in RQ1, and measured effectiveness in terms of MRR and PCR.

3.4.4.2 Results

As shown in Table 3.4, each module impacts the effectiveness of LAFF. The local modeling module improves the ability of BNs in ranking the correct values ahead of the incorrect ones, leading to a higher MRR value (while the PCR value remains equal to 1). The endorsing module mainly reduces the quantity of inaccurate suggestions made by different BN models (and therefore reduces the PCR value).

When we compare LAFF (with both modules enabled) with a plain BN (i.e., LAFF-LE) on the NCBI dataset, LAFF improves the MRR value by $+20$ pp (0.74 vs 0.54) for the sequential filling scenario and by $+17$ pp (0.73 vs 0.56) for the random filling scenario; on the PROP dataset the improvement is smaller ($+4$ pp for both scenarios). Hence, the integration of the local modeling and endorsing modules positively affect the effectiveness of LAFF. When we apply the endorsing module on a plain BN, we get an MRR improvement ranging from $+1\,\mathrm{pp}$ to $+5\,\mathrm{pp}$ on the two datasets (see LAFF-LE vs LAFF-L). Even though the local modeling module alone does not affect the number of suggestions, the integration of the local modeling in LAFF-L leads to a further reduction of the number of inaccurate suggestions (with PCR dropping by $0\,\mathrm{pp}$ to $13\,\mathrm{pp}$); it improves the MRR value by $0\,\mathrm{pp}$ to $+19\,\mathrm{pp}$ on the two datasets (see LAFF-L vs LAFF).

These results can be explained as follows. As mentioned in section 3.3.3, the endorsing module endorses suggestions based on two heuristics: $checkDep$ (which checks if the filled fields are parents of the target field) and $sumProb$ (which checks whether the sum of probabilities for the top-n_r suggested values is higher than a threshold). When the local modeling module is enabled, the local models can learn fine-grained dependencies for a field and exclude some useless dependencies that are present in a plain BN. As a result of the absence of the dependencies between filled fields and the target field, the $checkDep$ heuristic may endorse more suggestions (i.e., the PCR value decreases from LAFF-L to LAFF), in order to retain high-confidence suggestions (i.e., the MRR value increases). On the PROP dataset, the improvement is not obvious because of the high quality of its input instances. As a proprietary dataset from the banking domain, the data quality division usually double-checks the data entry to minimize the effect of data errors and data conflicts on financial software systems. Compared to a public dataset, BNs trained on the PROP dataset can find more meaningful field dependencies and also achieve high $sumProb$ values for most suggestions; hence the differences in both MRR and PCR values are relatively small when different modules are enabled.

The answer to RQ3 is that the local modeling module and the endorsing module improve the effectiveness of LAFF.

3.4.5 Impact of the Number of Filled Fields (RQ4)

LAFF takes as input a set of already-filled fields with their values to suggest the value of the target field. To answer RQ4, we assessed the impact of the number of filled fields on the effectiveness of LAFF as well as of the ARM and MFM baselines. We did not compare to the NaïveDT and FLS baselines. NaïveDT is impractical to be used in a production-grade system due to the timeout issue during the training phase. FLS does not use the information contained in the already-filled fields.

3.4.5.1 Methodology

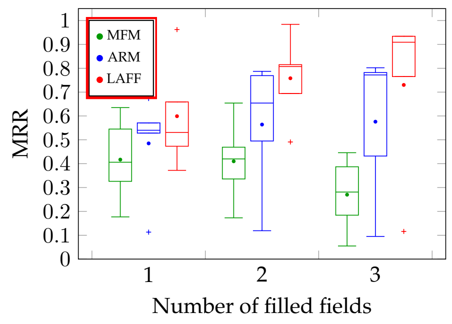
We generated new test sets by varying the number of filled fields on the testing input instances obtained as described in section 3.4.1. To generate a test set with i filled fields for a target t, for each testing input instance, we set the field t as the target and randomly selected i non-empty fields as filled fields. The unselected fields were considered as unfilled and their values replaced with a dummy value representing empty fields. Given a data entry form with t targets for automated form filling, we can generate t new test sets with i filled fields, each of which has a different target. We ran LAFF on these new test sets and computed the MRR and PCR values for predicting different targets. The results indicate the effectiveness of LAFF when i fields are filled.

Following this strategy, we assessed the effectiveness of LAFF and the baselines on the NCBI dataset with one, two, and three filled fields. We discarded the configuration with four filled fields, since we could only generate 505 new testing input instances due to the optional fields; this number is significantly smaller than the number of testing input instances we obtained for the configurations with one/two/three filled fields (which have more than 17 000 new testing input instances) and might have introduced bias.

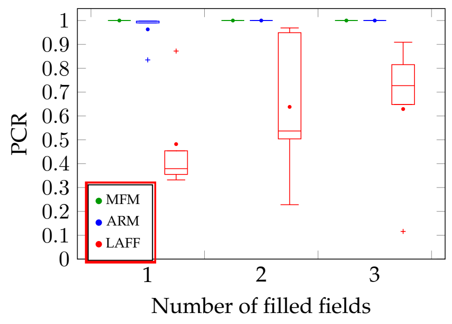
For the PROP dataset, we generated 16 test sets representing the configurations with one to 16 filled fields; each of them has more than 300 000 testing input instances. However, running the experiment for all testing input instances would be infeasible on the dedicated server provided by our industrial partner, which caps the duration of any job to 168 hours. The issue of the execution time was mainly introduced when evaluating ARM. As mentioned in section 3.4.3, ARM may take more than 3 s to provide a suggestion. This means ARM could take, in the worst case, $\approx 300\,000 \times 16 \times 3\,\mathrm{s} \approx 166$ days to execute on the testing input instances for all the 16 generated test sets in the PROP dataset. Hence, to assess the impact of the number of filled fields on all the algorithms when using the PROP dataset, we randomly sampled $12\,600$ testing input instances from each of the 16 newly generated test sets. We chose this number because, in the worst case, the experiments for all the algorithms could be finished within the 168 hours limit (e.g., $12\,600 \times 16 \times 3\,\mathrm{s} = 168$ hours for ARM).

3.4.5.2 Results

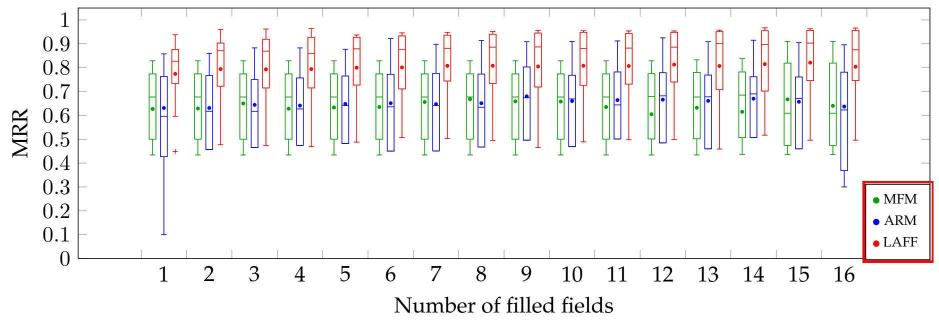
Figure 3.6 shows the results of running the form filling algorithms on the NCBI and PROP datasets with different numbers of filled fields. The x-axis



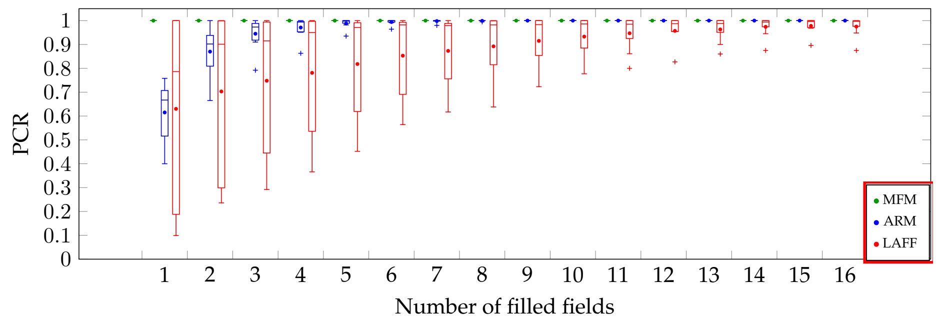
(a) MRR on the NCBI test sets



(b) PCR on the NCBI test sets



(c) MRR on the sampled PROP test sets



(d) PCR on the sampled PROP test sets

Figure 3.6: Effectiveness of LAFF with different number of filled fields on the NCBI and PROP datasets

of the figures represents the number of filled fields and the y-axis shows the MRR or PCR value of each form filling algorithm.

In terms of accuracy (in Figure 3.6a and Figure 3.6c), the MRR values of LAFF and ARM increase when more fields are filled. For example, the average MRR value of LAFF increases from 0.599±0.227 to 0.730±0.350 on the NCBI dataset and from 0.774±0.160 to 0.804±0.190 on the PROP dataset as the number of filled fields increases from one to three and from one to 16, respectively. In contrast, MFM shows a different trend. In theory, MFM is not affected by the number of filled fields, as it always provides the same suggestion for a target field, based on the most frequent values filled in the target field in the past. However, our experiments show a variation in MRR as the number of fields increases. More specifically, since the testing input instances generated for each number of filled fields are different, the MRR value of MFM decreases when the correct values for a target in the generated testing input instances are not the most frequent historical values. Overall, according to the boxplots, when varying the number of filled fields, the MRR values of LAFF for predicting different targets remain higher than those of MFM and ARM. Given a partially filled form with only one filled field, LAFF outperforms MFM and ARM by $+12$ pp and $+11$ pp, respectively, on the NCBI dataset; it also outperforms both MFM and ARM by $+14$ pp on the PROP dataset.

As for coverage, the PCR value of the baselines is 1 for the majority of the numbers of filled fields (as shown in Figure 3.6b and Figure 3.6d). For LAFF, the PCR value increases as more fields are filled. However, we find that the PCR values of LAFF for different target fields vary significantly, especially with a small number of filled fields. For example, with one filled field, the PCR values of LAFF are 0.482±0.22 and 0.630±0.41 respectively on the NCBI and PROP datasets. This is because, when the number of filled fields is small, the filled fields may not always provide enough knowledge (i.e., the information of dependent field values) for LAFF to predict all the target fields. For some targets, the endorsing module of LAFF may filter out many suggestions. When the number of filled fields increases, LAFF gets more information from the input instances to provide suggestions that can be endorsed. For example, the PCR value of LAFF increases to 0.975±0.04 on the PROP dataset with 16 filled fields. Overall, The PCR values show that LAFF could correctly endorse the suggestions when the form is partially filled, which helps LAFF achieve higher MRR values than those of the baselines.

The answer to RQ4 is that the effectiveness (in terms of MRR and PCR) of LAFF increases as more fields are filled. Further, LAFF can better handle partially *filled forms than state-of-the-art algorithms.*

3.4.6 Impact of the Size of the Training Set (RQ5)

LAFF is a learning-based approach that requires a training set (i.e., historical input instances) to train machine learning models. To answer RQ5, we assessed the impact of the size of the training set on the effectiveness of LAFF.

3.4.6.1 Methodology

We evaluated LAFF by varying the size of the training set from $10\,\%$ to $100\,\%$ of the historical input instances included in the training set, with a step of 10%. For the NCBI dataset, the size of the sampled training set ranged from 5928 to $59\,284$ historical input instances (where $59\,284$ is 80% of the $74\,105$ input instances in the dataset). For the PROP dataset, the size of the sampled training set ranged from $13\,955$ to $139\,557$ historical input instances (where $139\,557$ is 80% of the $174\,446$ input instances in the dataset).

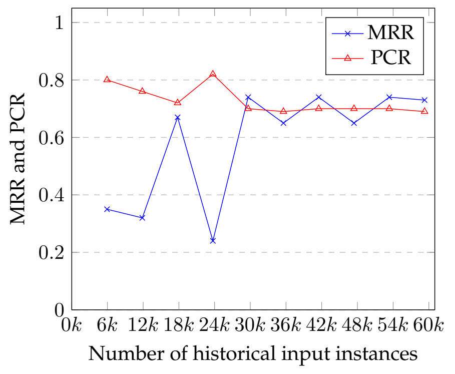
For a given percentage value p, we randomly sampled p% of the historical input instances in the training set to form a smaller training set. We trained LAFF on the sampled training set and used the trained model to conduct automated form filling on the testing input instances obtained as described in section 3.4.1.

As for RQ1, we measured the effectiveness of LAFF in terms of MRR and PCR.

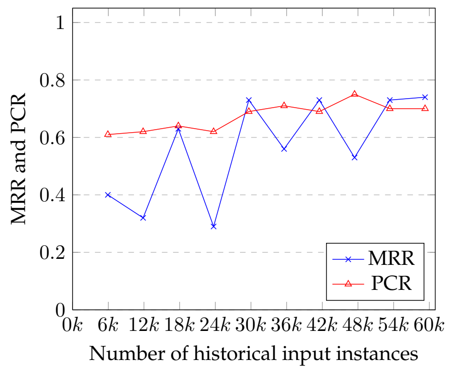
3.4.6.2 Results

Figure 3.7 shows the results of LAFF on the NCBI and PROP datasets with different training set sizes. The x-axis of the figure represents the number of historical input instances; the y-axis shows the MRR and PCR values of LAFF under different filling scenarios.

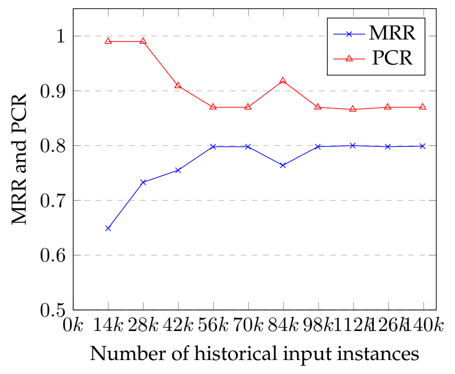
As shown in the figure, the value of MRR increases and gradually becomes stable on the two datasets when the size of the training set increases. For the NCBI dataset, the value of MRR is low and fluctuates significantly with a training set of less than $30\,000$ historical input instances; the majority of MRR values are lower than 0.40 for the two form filling scenarios. When we have more historical input instances (between $30\,000$ and $60\,000$), the value of MRR increases and becomes more stable, ranging from 0.650 to 0.740 for the sequential filling scenario and from 0.530 to 0.740 for the random filling scenario. The value of MRR has a similar trend on the PROP



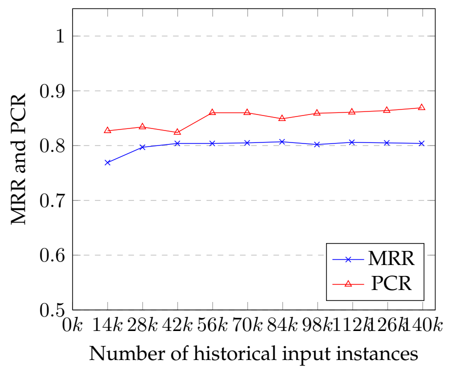
(a) Sequential filling scenario on NCBI dataset



(b) Random filling scenario on NCBI dataset



(c) Sequential filling scenario on PROP dataset



(d) Random filling scenario on PROP dataset

Figure 3.7: Effectiveness of LAFF with different training set sizes on the NCBI and PROP datasets

dataset; it gradually increases and then becomes stable (between 0.764 and 0.807) when the number of historical input instances is higher than 56 000.

In terms of *PCR*, the endorser module of LAFF works poorly with a small training set. LAFF either keeps the majority of suggestions (for the sequential filling scenario) or wrongly removes many suggestions (for the random filling scenario), leading to low *MRR* values. As the size of the training set increases to more than 30 000 (for the NCBI dataset) and 56 000 (for the PROP dataset), LAFF is able to remove more inaccurate suggestions, achieving a *PCR* value between 0.690 and 0.750 and between 0.849 and 0.918, respectively on the two datasets.

When comparing the results of LAFF across the two datasets, we observe a more significant fluctuation of MRR values on the NCBI dataset than that on the PROP dataset as the size of the training set increases. This is caused by the data quality of the NCBI dataset. In contrast to the proprietary dataset PROP, there is no field constraint or additional check on the values filled in the NCBI data entry form. This means that, when more historical input instances are added to the training set, one may also introduce many conflicting or erroneous field values, which increase the uncertainty of LAFF when predicting on the NCBI dataset.

The answer to RQ5 is that the size of the training set affects the effectiveness of LAFF. MRR values increase on both datasets when the size of the training set increases; more suggestions are also correctly endorsed. With more than 56 000 *historical input instances, LAFF achieves accurate suggestions on both datasets*

3.4.7 Threats to Validity

The size of the pool of historical input instances can affect the effectiveness of LAFF, a common issue among learning algorithms. Nevertheless, we do not expect this to be a strong limitation since it targets data entry functionalities in enterprise software, in which one can expect thousands of input instances per day, as it is the case for the system used by our industrial partner.

Another threat to the validity is the choice of the value of the endorser threshold θ. With a higher threshold, LAFF will filter out more suggestions (resulting in a lower PCR value), only keeping the ones with a high predicted probability (resulting in higher MRR value). Hence, this threshold reflects the degree of uncertainty users are willing to accept regarding the suggestions provided by LAFF. To mitigate this threat, we selected a threshold value based on the feedback received by data entry operators and data quality engineers of our partner.

The choice of the deployment method for LAFF can impact its performance in terms of prediction time. In our experiments, we deployed LAFF locally; using a different deployment (e.g., cloud-based) could lead to different results, since the prediction time would be affected by many other factors, such as the DNS lookup time, the connection time, and the data transmission time [CCY00]. Since the prediction time of LAFF is less than 317 ms, an application using a non-local deployment would have enough leeway to optimize these factors and provide seamless interactions for users, complying with human-computer interaction principles [Hee00] (i.e., with a response time less than 1 s). As part of future work, we plan to assess the performance of LAFF under different deployment configurations.

To increase the external validity of our results, LAFF should be further studied on other datasets, possibly from other domains. To partially mitigate this threat, we use two different types of datasets to evaluate LAFF, including both a public and a proprietary dataset, and the corresponding data entry forms. Meanwhile, we simulated two form filling scenarios (sequential and random filling) that are plausible during a real data-entry session. As part of future work, we plan to conduct a user study using different datasets and data entry forms, to analyze the effect of LAFF on reducing form filling time and input errors. Another external threat is our implementation of the four algorithms (MFM, ARM, NaïveDT, and FLS) to which we compared, which may be different from the original definitions; to mitigate this threat, we cross-reviewed the implementations, taking into account the relevant literature (when available).

3.4.8 Data Availability

The implementation of LAFF, the NCBI dataset, and the scripts used for the evaluation are available at https://figshare.com/s/b191fcb5ee7f3ad634bf; LAFF is distributed under the MIT license. The PROP dataset cannot be distributed due to an NDA.

3.5 Discussion

3.5.1 Usefulness

The fundamental question we seek to answer is whether LAFF can help users fill forms. To answer this question, we evaluated LAFF with two real-world form filling datasets, one from the biomedical domain and another from the banking domain. The results show that LAFF outperforms state-of-the-art form filling algorithms in providing a larger number (with a PCR value over 0.70) of accurate suggestions (with a MRR value over 0.73). The MRR value reflects the ability of LAFF in avoiding inaccurate suggestions. For example, considering a list with three suggested values, if the correct value is in the top-1, top-2, and top-3 of the list, the corresponding MRR value is 1, 0.5, and 0.33, respectively; when all the values are incorrect, the MRR value is 0. In the context of our experiments, an MRR value of at least 0.73 indicates that, for more than 73% of suggestions, LAFF can help users find the correct value from the top-ranked ones. The PCR value indicates the number of suggestions made by LAFF. A PCR value over 0.70 indicates that LAFF can confidently make suggestions for more than 70% of target fields, where the correct values are usually ranked before the incorrect ones. According to a previous study [CCC+11], as users are presented with more candidate values for selection, they tend to make more mistakes, which makes form filling a frustrating activity. Hence, we speculate LAFF reduces the mental load of users in filling forms by helping them go through fewer candidate values (top-k%) before finding the correct one; further user studies are required to corroborate this hypothesis.

We can interpret the above results from a point of view of usefulness as follows. On the one hand, as shown in RQ1, with an MRR value over 0.73, $79.0\,\%$ to $90.1\,\%$ of the suggestions made by LAFF allow users to find the correct value among the 5% top-ranked suggested values on the two datasets. This means that when LAFF makes suggestions, in $79.0\,\%$ to $90.1\,\%$ of the cases, it leads to at least 95% effort saving when browsing possible values, since users need only to check the top 5% most likely items recommended by LAFF. For example, on our datasets, users need to check between one (for field "activity" having 13 candidate values) and ten (for field "country" having 206 candidate values) values before finding the correct one. On the other hand, according to our analysis in RQ1, a frequency-based method like MFM achieves MRR values ranging from 0.42 to 0.64 on the NCBI and PROP datasets. Hence, when comparing LAFF with the widely-used data entry solution MFM, an MRR improvement of $+14$ pp to $+32$ pp suggests that LAFF can better help users select candidate values in data entry forms.

3.5.2 Practical Implications

This subsection discusses the practical implications of LAFF for its different stakeholders: software developers, end users, system administrators, and researchers.

3.5.2.1 Software Developers

Automated form filling is a common requirement for software systems. Some popular languages and APIs (e.g., HTML [W3C21] and Android APIs [And21]) also provide pre-fill or auto-completion frameworks, for which customized form filling strategies can be implemented. In its current version, LAFF is a stand-alone tool that developers can integrate into their data entry form implementations as an effective and efficient strategy for filling categorical fields (e.g., Listboxes and Dropdown Lists). Since LAFF shows a higher accuracy than state-of-the-art approaches, it can be used in data-reliant enterprise systems across different domains, especially when there are constraints on sharing or accessing data of other software systems.

As for adopting LAFF in production, in terms of execution time, a training time of about one hour allows LAFF to compute daily updates for its models, empowered with the information derived from new input instances. Moreover, considering the interactive nature of data-entry applications, we use a model selection strategy to select the suitable local or global model for prediction from multiple models. Compared with many learning strategies that consider the predictions of multiple models jointly (e.g., ensemble learning), our strategy could significantly reduce prediction time in specific deployment configurations. For example, the worst-case prediction time of ensemble learning is, when using a single-thread configuration, the sum of the prediction time of all local models. This could be reduced to the maximum of the prediction time of all local models by using multiple threads in parallel. This strategy is viable when the trained models are deployed on a powerful server, and the prediction is performed server-side. However, some enterprises (as it is the case for our industrial partner) might prefer to integrate a form filling approach like LAFF *locally*, on the machines used by data entry operators, to avoid network latency during data entry sessions. In such a deployment configuration, using multiple threads could significantly slow down the data entry session, especially on machines with a limited computing power. For these reasons, we designed LAFF with a lightweight model selection strategy, instead of using alternatives like ensemble learning.

Integrating LAFF into a data entry form requires providing a mapping between the field names and the column names in the dataset. This mapping needs to be identified only once and therefore does not have much impact on the practical application of LAFF. We remark that, in modern applications built using the MVC pattern for the UI and an Object-Relational Mapping (ORM) framework for data persistence [CSY+16], the mapping can be retrieved from the ORM framework configuration. Other sources for the mapping can be software design documentation, such as the database schema and the description of the UI widgets in the data entry forms.

3.5.2.2 End Users

Form filling is time-consuming and error-prone for end users, which can cause more than half of data errors [QMR+20]. These errors seriously affect data-reliant software systems and even cause loss of human life [Ame05, QMR+20, KJ10, Ban03]. In this study, we have proposed LAFF to improve the accuracy and efficiency of the data entry process executed by end users, when filling categorical fields. First, LAFF uses an endorser module to alleviate the cognitive load on users caused by wrong suggestions; this module significantly improves the accuracy of suggestions. Second, LAFF helps users focus on the most-likely candidate values in a list of values. It decreases the number of candidate values users need to browse. *We follow the practice of designing an effective recommender system* (e.g., a form filling system) in software engineering, which is to avoid "helpless" suggestions to be ignored by users, but provide them with a large number of "helpful" suggestions [RWZ09].

Furthermore, in the experiments, LAFF can provide suggestions within at most 317 ms for each target field, which enables end users to conduct seamless interaction with the data entry form.

3.5.2.3 System Administrators

The deployment of LAFF in production requires system administrators to configure its parameters. This can be achieved with the help of domain experts, based on their domain knowledge. A group of configurable parameters is set in the pre-processing step. We implement this step based on best practices for predictive data mining [AKV19]. A threshold that can affect auto-filling effectiveness is θ, for the endorser module. This threshold reflects how much uncertainty domain experts are willing to accept regarding the suggestions provided by LAFF. Such configuration allows domain experts to use LAFF according to their requirements and application scenarios.

Overall, the configuration parameters represent the only domain-specific aspect of LAFF. Everything else about it is domain-agnostic. Deploying LAFF for a new form only requires to collect the input instances of the form for a certain period of time and use these instances to train the form filling models.

3.5.2.4 Researchers

In this chapter, we use a local modeling module to effectively learn the fine-grained dependencies on historical input instances. Since existing approaches in software engineering perform local modeling on numerical data instances (e.g., software metrics [MBM$^+$11]), we propose a novel solution to solve the problem of using local modeling on categorical data instances. We speculate that our proposed solution can inspire the adoption of local modeling for different data types in many software engineering tasks. In addition, we use an endorser module to decide if suggestions are accurate enough to be provided to end users. Such a module is important for algorithms where 100% accuracy cannot be achieved in practice. For example, during form filling, predicting all the other fields based on only one filled field may lead to inaccurate suggestions. In this case, it is more practical to automatically remove inaccurate suggestions using an endorser module. We believe the endorser-based architecture discussed in section 3.3.3 can be adopted by other recommender systems.

Furthermore, the error analysis for RQ1 described in section 3.4.2 suggests possible research directions, such as learning from users' corrections and using multi-objective optimization for form refactoring (with a new order of fields) for improving form filling suggestions.

3.5.3 Limitations

3.5.3.1 Type of Fields

In this work, we have focused on predicting the values of categorical fields, based on the historical information available from the same software system; LAFF does not work with other types of fields. Due to the unique characteristics of each type of field, distinct solutions have been proposed in the literature (e.g., text auto-completion for textual fields, data testing for numerical fields). However, these solutions cannot help users fill categorical fields, which is a critical task during form filling. We acknowledge that other types of fields may contain critical data that could cause major problems. However, empirical studies show that selection errors lead to more than half (54.5%) of the data errors in a software system [QMR+20]. For example, the selection of wrong drugs [KJ10] or the wrong modality of care [QMR+20] in medical record systems can even cause loss of human life. In addition, as shown in the NCBI and PROP datasets (in section 3.4.1), the majority of categorical fields we evaluated (with the number of candidate values ranging from 13 to 208) are related to certain domains or business processes. These fields are more difficult to fill than fields such as "sex" and "age", since users need to understand the meaning of candidate values. Errors in these fields can cause significant business problems. For example, the selection of a wrong "field of activity" when opening a bank account may cause business loss between the company and the bank. By knowing the actual "field of activity" of the company, the bank could have offered targeted products to its customer since the beginning of the business relation. Hence, there is a need for a semi-automated method that supports and guides users when filling categorical fields.

3.5.3.2 Cognitive Load on Users

The suggestions made by automated form filling tools like LAFF can increase the cognitive load on users, when the tool provides a long list of suggested values/options for users to check or when the suggestions do not include the correct value. To reduce the cognitive load in the first case, we configured LAFF to suggest the top 5% most likely candidate values for each target field, rather than reordering all the candidate values based on their probability. Although LAFF does not directly reduce the number of candidate values in a field, it highlights the top 5% values, on which users can focus. As for the second case of cognitive load (suggestions not including the correct value), LAFF includes an endorser module to decide whether the suggestions are accurate enough to be returned to users. As shown in Table 3.4 and as part of the answer to RQ3, the endorser module significantly reduces the number of possibly inaccurate suggestions while increasing the MRR values. As a result, our experiments show that $79.0\,\%$ to $90.1\,\%$ of the LAFF suggestions allow users to find the correct value among the top 5% suggested values.

3.5.3.3 Fields with Semantic Overlap

Candidate values in categorical fields could have semantic overlaps. For example, let us consider the case of field "field of activity" with two possible values, "banking service" and "financial services". From a semantic point of view, the former is a specific case of the latter. This form of semantic overlap affects LAFF as follows. During the model building phase, LAFF could be trained with inconsistent historical input instances (e.g., two historical instances that have same values in all fields but "field of activity", with one instance having "banking service" and the other "financial services"). In this case, LAFF cannot build, with enough confidence, dependencies between the values in the field affected by a semantic overlap (e.g., "field of activity") and the other fields. During the form filling suggestion phase, when LAFF identifies both values as candidate values to suggest (e.g., with similar probability), due to the endorser module, LAFF may not provide any suggestions or suggest both values to users (depending on the number of suggested values n_r and the threshold θ). In both cases, users need to decide which value to select by themselves. However, defining candidate values with semantic overlaps is not a good practice for form design, since it increases the mental load on users to decide which value is more appropriate (e.g., the more specialized "banking service" or the more generic "financial service"). We suggest to address this issue by refining the candidate values during the design phase of the system.

3.5.3.4 Cases of Limited Accuracy

As discussed in the "Error analysis" part of the answer to RQ1 (page 40), there are $9.9\,\%$ to $21\,\%$ of suggestions made by LAFF for which the correct value is not in the top 5% suggested values. We have identified two main reasons leading to low accuracy: LAFF tends to provide incorrect suggestions, when (1) the number of filled fields used for prediction is small, and (2) the size of the training input instances for a target field is small. We plan to address these limitations as part of future work, along the lines mentioned on page 40.

3.5.3.5 Learning from Corrections

Automated form filling is an interactive process between users and the automated tool. In the current version, LAFF does not offer the possibility, during a data entry session, for a user to correct LAFF's suggestions by selecting a candidate value that is not presented in the top 5% suggested values. If a user makes many corrections to suggestions during a data entry session, it means that the learned probability table (BN) may not reflect the relations of values in the current input instance. The ability to learn from these users' corrections could further improve the accuracy of LAFF. One intuitive solution is to assign a higher weight on such input instances when re-training LAFF for learning new relations. For example, input instances with many inaccurate suggestions can be oversampled to increase their proportion in the entire historical of input instances.

3.5.3.6 Cold Start

LAFF depends on the dataset associated with an input form; it always has to be trained for a new system. The size of the pool of historical input instances can affect the accuracy of LAFF, a common issue among learning algorithms. When there are no or only a few historical input instances (i.e., the cold start problem [LIJ+19]), the accuracy of LAFF is limited. As presented in RQ5 (in section 3.4.6), MRR values of LAFF increase when the size of the training set increases; LAFF achieves accurate suggestions with $30\,000$ and $56\,000$ historical input instances on the NCBI and PROP datasets, respectively. Nevertheless, we do not expect the size of the training set to be a strong limitation on the feasibility of applying LAFF. First, LAFF targets data entry functionalities in enterprise software, in which one can expect thousands of input instances per day. For example, nowadays the NCBI platform gets about 9600 input instances per month for a single species (i.e., "Homo sapiens"). It is also the case for the system used by our industrial partner. Second, LAFF does not require additional effort to label training data; it uses the actual values filled by users as the ground truth to train the model. With an adequate training set size, LAFF can directly provide accurate suggestions for users.

3.6 Related Work

The approach proposed in this chapter is mainly related to works on automated form filling based on the information from *the same software system*, which focuses on filling free-text fields and categorical fields. Regarding the former, the main proposals use language models (e.g., n-gram and sequence-to-sequence learning) to learn relationships between characters or words from historical textual inputs [VDBB08, SAM18]; these relationships are then used to provide word auto-completion based on the letters typed in a field [ZZW19]. As for dealing with categorical fields, most of the approaches suggest possible values from a list of candidate values. Martínez-Romero et al. [MROE$^+$19] use association rule mining (ARM) to uncover the hidden associations of fields for real-time form filling; however, as shown in our experiments, ARM does not provide accurate suggestions when compared with LAFF. Hermens and Shlimmer [HS94] applied decision tree and hierarchical clustering for filling forms in an electronic leave report system: the study reports that these algorithms performed worse than the simple most-frequent method when the forms were filled out in a random order. Troiano et al. [TBA17] trained a Bayesian network model on the historical inputs from a user for an online payment system. The trained model could auto-fill the payment form for the same user by reusing his historical payment records. The plain Bayesian network model is also trained by Ali and Meek [AM09] for auto-filling the bug submission form of a bug tracking system. Compared to existing approaches, LAFF performs automated form filling by mining field dependencies on input instances from different users. We proposed local modeling and a heuristic-based endorser to further improve the accuracy of form filling suggestions.

Automated form filling has also been investigated in the context of developing "smart" personal information management systems, to support information exchange *across software systems* [FGG$^+$12, WGV$^+$11, AGLH10, CFM02]. In this context, the main challenge is the semantic mapping of fields across software systems, e.g., how to map the "postal code" and "zip code" fields (from two different software systems) to the same concept. To address this challenge, Chusho et al. [CFM02] manually construct rules to merge similar concepts of commonly used field names. Other works use string-based matching [HCH04], WordNet [AGLH10], and Wikipedia [HM09] as additional resources to calculate the similarity among fields. Wang et al. [WZK+14] calculate field similarity based on the field names, form topics, and names of neighbor fields. They recently propose to use learning-to-rank algorithms to further improve form filling effectiveness [WZNN17]. However, to learn mapping rules, these algorithms require access to the users' personal records from different software systems. In contrast, we only use input instances from the targeted software system; thus, our approach can be used when there are legal or security constraints on sharing records across systems [WGV+11]. Moreover, the above approaches perform well only on common fields like "age" and "address", and cannot cope with fields that are domain-specific and used only in few software systems.

The filling order of fields influences the ability of form filling algorithms [HS94]. Several works refactor data entry forms to provide effective supports for form filling, for example using fields dependencies [CCC+11, TBA17] or user roles [ABY16]. All these approaches identify the fields that a target field directly depends on and change their (field) order so that they can be filled before the target field, to increase the accuracy of predicting the latter. Form refactoring can be regarded as a preliminary step to our proposed approach.

Form filling can be considered as a task to auto-complete data entry for software systems, which in the literature is a popular research topic in many domains; however, most of the existing approaches propose auto-completion for different purposes or with different inputs than what we considered in this chapter. In software engineering, several techniques have been proposed to provide suggestions for certain fields of software systems (e.g., priority of bug reports in bug tracking systems [ULI19] and elements of a domain model when using domain-model designing systems [BCL+21]). Suggestions are provided by analyzing the textual data (e.g., bug reports, requirement documents) with natural language processing techniques; instead, in our work we analyze the dependencies of categorical fields. In the field of information retrieval, crawlers automatically fill and submit web forms to crawl the data returned from databases [HRR19, KMH15]; in the context of data crawling, automated form filling aims to automatically generate input values that can pass the field validation and retrieve more data, instead of helping users find the correct value they intend to fill. In the field of data mining, several approaches [TCdSdM10, KCVM04, DOP13] fill data entry forms with the metadata extracted from data-rich text files. For example, these approaches can

Table 3.5: Overview of related work

Scenario	Reference	Task/Example	Difference with LAFF
Form filling under the same software system	Salama et al. [SAM18] Van Den Bosch and Bogers [VDBB08] Zhang et al. [ZZW19]	**[Task]** Predict or auto-complete the next character or word for textual inputs. **[Example]** They suggest *Angeles* when users type the word *Los*.	They build language models on the values in *textual fields* instead of categorical fields; the user's initial input is required.
	Martínez-Romero et al. [MROE+19] Hermens and Shlimmer [HS94] Troiano et al. [TBA17] Ali and Meek [AM09]	**[Task]** Suggest the correct value from a list of candidate values in *categorical fields*. **[Example]** They suggest *Leasing Service* from a list of options in the field "primary field of activity".	We compared LAFF with algorithms MFM [HS94], ARM [MROE+19], and NaïveDT [HS94] as part of RQ1; we compared to a plain BN (equivalent to the one proposed in [TBA17] and [AM09]) as part of RQ3.
Form filling across software systems	Araujo et al. [AGLH10] Chusho et al. [CFM02] Firmenich et al. [FGG+12] Winckler et al. [WGV+11] He et al. [HCH04] Hartmann and Muhlhauser [HM09] Wang et al. [WZK+14] Wang et al. [WZNN17]	**[Task]** Prefill form fields by reusing the values filled in other web forms; they build the mapping or ontology of field names across web forms. **[Example]** They use the value filled in "postal code" in one web form to help the same user fill "zip code" in another web form.	These approaches cannot be applied to fields that are domain-specific and used only in few software systems (e.g., medical systems); enterprise information system may have constraints on visiting or sharing records across systems. LAFF is designed to work in these scenarios.
Form refactoring	Chen et al. [CCC+11] Troiano et al. [TBA17] Akiki et al. [ABY16]	**[Task]** Re-order fields to support effective form filling. **[Example]** They refactor the field "role" as the first field, if they find "role" is informative to predict values of other fields.	It is a preliminary step, complementary to automated form filling
Software artifacts information auto-completion	Umer et al. [ULI19] Burgueño et al. [BCL+21]	**[Task]** Predict values of certain fields in software systems. **[Example]** They predict "bug priority" based on the description of bug reports.	It relies on textual data (e.g., bug reports and requirement documents) for suggestions. LAFF does not rely on such information.
Data crawling	Hernández et al. [HRR19] Kantorski et al. [KMH15]	**[Task]** Crawl data from databases by form filling. **[Example]** They generate value combinations for fields in a job search site to crawl all the job information hidden in the database.	They aim to generate valid input values for data crawling, instead of helping users find the correct candidate value.
Data mining	Diaz et al. [DOP13] Kristjansson et al. [KCVM04] Toda et al.[TCdSdM10]	**[Task]** Extract data values from data-rich text to fill forms. **[Example]** They use a resume text file to fill several fields of forms in job search sites.	They rely on the extraction of information in data-rich files (e.g., text files and spreadsheets). LAFF does not rely on such information.

automatically use the metadata taken from a resume text file to fill several fields of forms in different job search sites [TCdSdM10]. However, in this study we do not rely on data-rich text files to infer field values.

Table 3.5 presents an overview of related work and their differences with LAFF.

3.7 Summary

In this chapter, we proposed LAFF, an approach to automatically suggest possible values of categorical fields in data entry forms, which are common user interface features in many software systems. Our approach utilizes Bayesian Networks to learn field dependencies from historical input instances. Moreover, LAFF relies on a clustering-based local modeling strategy to mine local field dependencies from partitions of historical input instances, to improve its learning ability. Furthermore, LAFF uses a heuristic-based endorser to ensure minimal accuracy for suggested values.

We evaluated LAFF by assessing its effectiveness and efficiency in form filling on two datasets, one of them proprietary from the banking domain. Evaluation results show that LAFF can provide a large number of accurate form filling suggestions, significantly outperforming state-of-the-art approaches in terms of Mean Reciprocal Rank (MRR). Further, LAFF takes at most 317 ms to provide a suggestion and is therefore applicable in practical data-entry scenarios.

Chapter 4

Learning-Based Relaxation of Completeness Requirements for Data Entry Forms

4.1 Overview

In this chapter , we propose LACQUER, a Learning-bAsed Completeness reQUirEments Relaxation approach, to automatically relax completeness requirements (i.e., relaxing a required field to be 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 [BLBB22], 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 4.5.

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 chapter 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.

The rest of the chapter is organized as follows. Section 4.2 provides a motivating example and explains the basic definitions of automated completeness requirements relaxation and its challenges. Section 4.3 describes the different steps and the core algorithms of LACQUER. Section 4.4 reports on the evaluation of LACQUER. Section 4.5 surveys related work. Section 4.6 discusses the usefulness and practical implication of LACQUER. Section 4.7 concludes the chapter.

4.2 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.

4.2.1 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. In this study, a field can only have one completeness requirement; in other words, a field cannot be optional and required at the same time. 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 [SHBA+14]) 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* [BBG11] or context-sensitive forms [AW12], 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 [AW12], 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 [VLL+03] — 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[1]. We plan to support this kind of complex controls as part of future work.

4.2.2 Motivating Example

Data entry forms are difficult to design [FGG+12] and subject to frequent changes [YSY+20]. 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 no-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 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*.

[1]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".

These meaningless values filled during form filling negatively affect data quality [AW12], 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 [AW12].
- **Error propagation**: Meaningless value errors can propagate and create more errors [MBM15], 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:

- **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 [WKL04, Ghe17], 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 [AW12] (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.

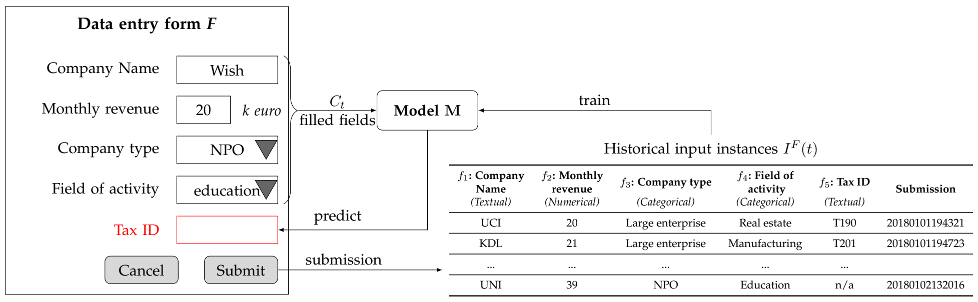
4.2.3 Problem Definition

In this chapter, 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 $V_i = VD_i \cup \perp$ if the field is optional ($\perp$ 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



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
...	...	...	...	...	
UNI	39	NPO	Education	n/a	20180102132016

Figure 4.1: The Automated Form Filling Relaxation Problem

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 in $I^F_{t_j}$ 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; $I^F(t) = \{I^F_{t_i}, I^F_{t_j}, \ldots, I^F_{t_k}\}$, 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 $I^F(t)$.

4.2.3.1 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.

4.2.3.2 Application to the running example

Figure 4.1 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., $F^r = \{f_1, f_2, f_3, f_4, f_5\}$). Values filled in these fields are then stored in a database. An example of the database is shown on the right side of Figure 4.1. 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., $I^F_{20180101194321} = \{\langle \textit{“Company Name”}, \textit{UCI}\rangle, \ldots, \langle \textit{“Tax ID”}, \textit{T190}\rangle\}$), 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 [BLBB22]. 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.

4.2.4 Towards adaptive forms: challenges

Several tools for adaptive forms have been proposed [FS98, BBG11, SLDM18]. These approaches use intermediate representations such as XML [BBG11] and dynamic condition response graphs [SLDM18] 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 require-

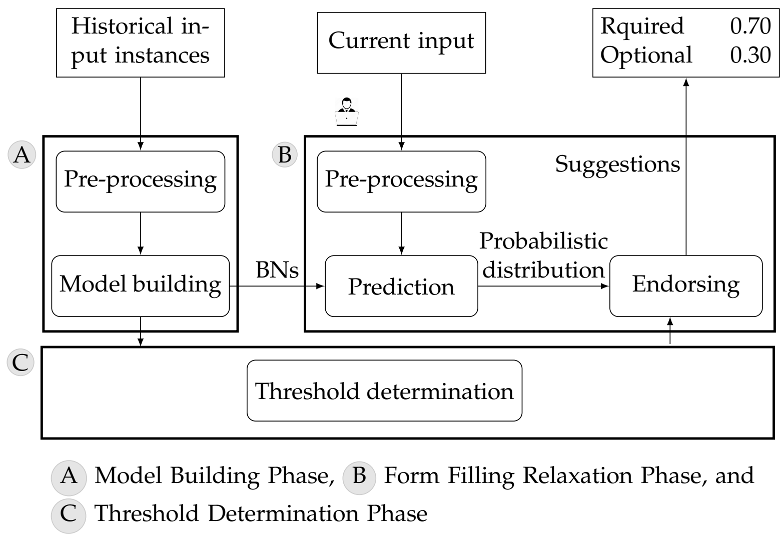

Historical input instances
Current input
Rquired 0.70
Optional 0.30
A
Pre-processing
Model building
BNs
B
Pre-processing
Suggestions
Probabilistic distribution
Prediction
Endorsing
C
Threshold determination
A Model Building Phase, B Form Filling Relaxation Phase, and
C Threshold Determination Phase

Figure 4.2: Main Steps of the LACQUER Approach

ments, 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 [FGG+12]. 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 [DVDSB+19, AZ17]. On the other hand, data entry forms are subject to change: a recent study [YSY+20] 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.

4.3 Approach

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

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→ Required	20→[20,22)	Large enterprise	Real estate	T190
KDL → Required	21→[20,22)	Large enterprise	Manufacturing	T201
EoP→ Required	@→Optional	Large enterprise	Manufacturing	T200
UNI→ Required	39→[39,41)	NPO	Education	n/a→Optional

Figure 4.3: Example of Pre-processed Historical Input Instances

As shown in Figure 4.2, 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 [BLBB22]; the main differences between the two approaches are discussed in section 4.5.

4.3.1 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 4.2.1, 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[2]; 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 [BFSO84]. 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 "Required" or "Optional", because such fields do not provide any discrimi-

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

native 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).

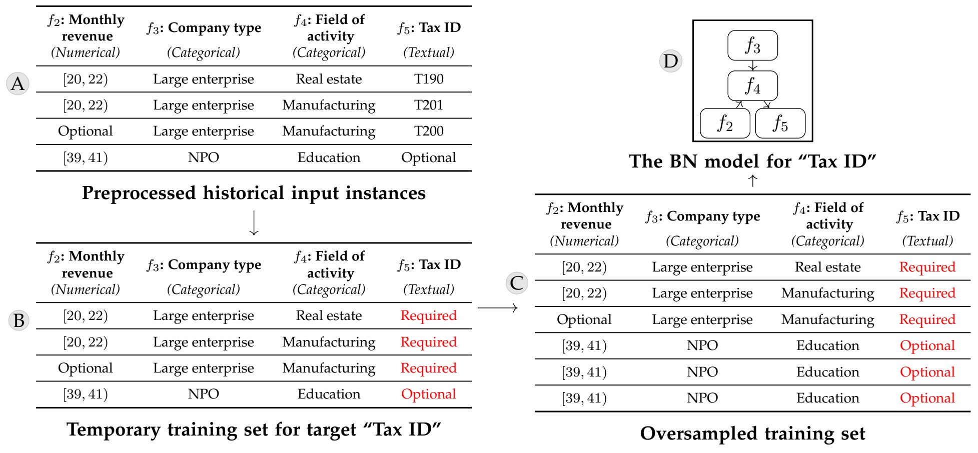
4.3.1.1 Application to the running example

Figure 4.3 shows an example of historical input instances collected from the data entry form presented in Figure 4.1. 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 4.1, 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)*.

4.3.2 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 [BLBB22] 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 [DSX10]



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

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"

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

Figure 4.4: Workflow of the Model Building Phase

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 [BLBB22], 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 2.1); BNs automatically deal with the missing conditionally independent variables (i.e., unfilled fields).

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 [BLBB22], we use hill climbing as the optimizer to learn the BN structure with a fitness function based on BIC [Raf95].

Algorithm 3 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 4.4). Initially, for each field f_i in the list of

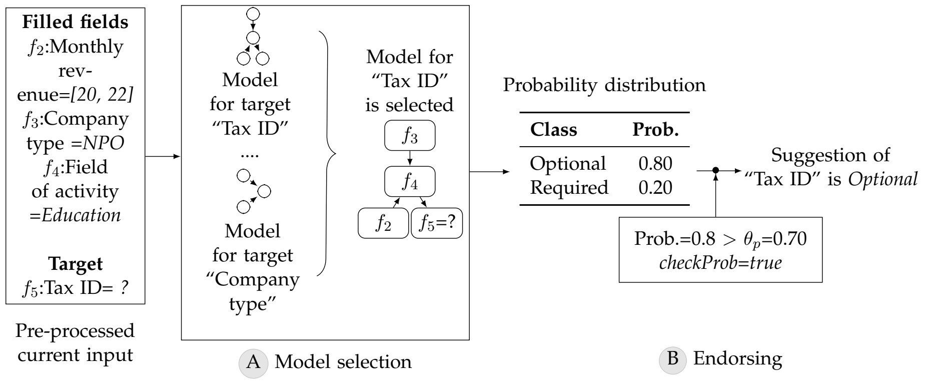
Algorithm 3: Model Building

```
Input: Set of preprocessed historical input instances $I^F(t)'_{train}$ for training
Output: Dictionary of probabilistic graphical models $\mathcal{M}$
1 $\mathcal{M}$ ←empty dictionary;
2 List of fields $fields \leftarrow getFields(I^F(t)'_{train})$;
3 foreach field $f_i \in fields$ do
4     Temporary training set $train_{f_i} \leftarrow getDataSetForField(I^F(t)'_{train}, f_i)$;
5     Oversampled Training set $train^{oversample}_{f_i} \leftarrow SMOTE(train_{f_i}, f_i)$;
6     Model $M_i \leftarrow trainBayesianNetwork(train^{oversample}_{f_i})$;
7     $\mathcal{M}[f_i] \leftarrow M_i$
8 end
9 return $\mathcal{M}$;
```

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 4.4). These two values are the classes according to which to predict f_i.

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 [JK19], since machine learning models may consider the minority class as noise [RA20]. The trained models could also over-classify the majority class due to its increased prior probability [JK19]. For example in block B of Figure 4.4, 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 $train^{oversample}_{f_i}$ (as shown in block C in Figure 4.4). 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 D in



Filled fields
f_2:Monthly revenue=*[20, 22]*
f_3:Company type =*NPO*
f_4:Field of activity =*Education*
Target
f_5:Tax ID= ?
Pre-processed current input
Model for target "Tax ID"
....
Model for target "Company type"
Model for "Tax ID" is selected
f_3
f_4
f_2
f_5=?
A Model selection
Probability distribution
Class | Prob.
Optional 0.80
Required 0.20
Prob.=0.8 > θ_p=0.70
checkProb=true
Suggestion of "Tax ID" is *Optional*
B Endorsing

Figure 4.5: Workflow for Form Relaxation Phase

Figure 4.4 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 3 is the dictionary $\mathcal{M}$.

4.3.2.1 Application to the running example

Given the preprocessed training input instances shown in block A in Figure 4.4, 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 B). 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 C of Figure 4.4). 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 D in Figure 4.4. After the model building phase, LACQUER outputs a model for each target. For the example of training input instances related to Figure 4.1, LACQUER returns five distinct models where each model captures the completeness requirement dependencies for a given target.

Algorithm 4: Form Filling Relaxation

```
Input: Dictionary of Models $\mathcal{M} = \{f_1 : M_1, \ldots, f_k : M_k\}$
       Set of Filled fields $F^f = \{\langle f_1^f, v_1^f \rangle, \ldots, \langle f_m^f, v_m^f \rangle\}$
       Target field $f_p$
       Threshold $\theta_p$
Output: A Boolean $checkOP_p$, representing the decision to set field $f_p$ to optional
1  Set $F^{f'} \leftarrow getPreprocessedData(F^f)$ ;
2  Model $M_p \leftarrow \mathcal{M}[f_p]$;
3  List of pairs of completeness requirements and probability distribution
     $\langle cr, pr \rangle_{list} = predictCR(M_p, F^{f'}, f_p)$;
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 $checkOP_p$;
```

4.3.3 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 F^f. The main steps are shown in Algorithm 4.

The inputs of the algorithm are the dictionary of trained models $\mathcal{M}$, the set F^f 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 § 4.3.1 to the set of the filled fields in F^f in order to obtain a new set of preprocessed filled fields $F^{f'}$ (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).

4.3.3.1 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 *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 § 4.3.4). 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 [WC21].

4.3.3.2 Application to the running example

Figure 4.5 shows the process of predicting the completeness requirement of the field "Tax ID" based on the input values in Figure 4.1. LACQUER first selects the model related to the current target for prediction (block A in

Algorithm 5: 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 $\theta$
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 $\theta$;
```

Figure 4.5). 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 B in Figure 4.5) 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.

4.3.4 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.

The main steps are shown in Algorithm 5. 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 4.4(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.

4.4 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?*

Table 4.1: 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)

4.4.1 Dataset and Settings

4.4.1.1 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 3.6) were either not mentioned [FS98, TB96], unavailable [BBG11], or confidential [SLDM18]. 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" [MK19]). 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 [BCG+12]. We choose this dataset because it has been used in previous work on automated form filling [MROE+19, BLBB22]. Moreover, this dataset provides the design of the corresponding data entry form in the CEDAR workbench [GOMR+17] with the list of completeness requirements for the different fields. Following the evaluation methodology described in previous work [MROE+19], 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[3]. The dataset is 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 values of the form fields as submitted by a user.

As shown in Table 4.1, the NCBI dataset contains 235 538 instances[4] 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 4.3.1, i.e., mapping the actual value in the data with the dictionary of meaningless values obtained from the domain knowledge. In Table 4.1, next to each field we indicate the ratio of instances having missing or meaningless values. The ratio of meaningless values[5] 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 4.1, 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 [SZZZ17]).

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

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

[4] The number of instances is different from that indicated in our previous work [BLBB22] 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.

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

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 4.1, 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 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.

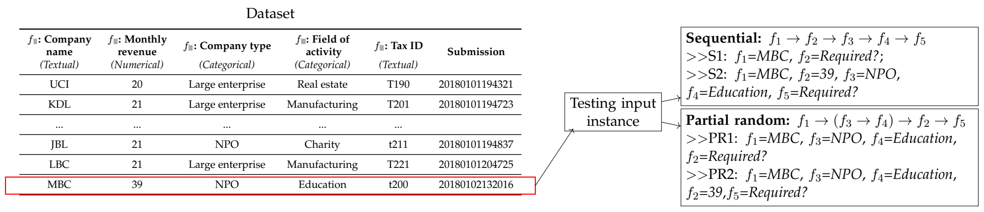
4.4.1.2 Dataset Preparation

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 4.3.4.

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 [FS04]. It simulates the logical order many users follow to fill out forms, especially when they use a keyboard to navigate form fields [Mic13]. 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 [CCC$^+$11]; 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)



Dataset

f▮: Company name (Textual)	f▮: Monthly revenue (Numerical)	f▮: Company type (Categorical)	f▮: Field of activity (Categorical)	f▮: 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?*

Figure 4.6: Example of Filling Orders

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.

4.4.1.3 Dataset Preparation - Application Example

Figure 4.6 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 4.2.3. 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 4.6) 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.

4.4.1.4 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[6], we performed them on a server running CentOS 7.8 on a 2.60 GHz Intel Xeon E5-2690 processor with 125 GB memory.

4.4.2 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) [MROE+19] 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 [Coh95]; it creates a rule set by progressively adding rules to an empty set until all the positive instances are covered [LVMPG14]. 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 [SGS18, GMH15]. Similar to ARM, Ripper suggests the consequent of the matched rule to users.

4.4.2.1 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

[6] 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).

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 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 [MROE$^+$19, BLBB22] in which it was applied in the context of form filling.

4.4.2.2 Results

Table 4.2 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

Table 4.2: 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

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 4.3.1), 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 a 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 LAFF 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).

4.4.3 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.

4.4.3.1 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.

4.4.3.2 Results

The results are presented in columns *Train* and *Predict* in Table 4.2. 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 4.2, 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 [Hee00], 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 [BLBB22]. 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 than 20 minutes and a prediction time of at most 839 ms *per target field, is suitable for practical application in data-entry scenarios.*

4.4.4 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 4.3.2), 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.

4.4.4.1 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 4.3 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.

Table 4.3: Effectiveness of LACQUER with Different Modules

ID	Module		NCBI								PEIS							
			Sequential				Partial Random				Sequential				Partial Random			
	S	E	Prec	Recall	NPV	Spec	Prec	Recall	NPV	Spec	Prec	Recall	NPV	Spec	Prec	Recall	NPV	Spec
LAFF-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
LAFF-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
LAFF-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
LAFF	✓	✓	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

4.4.4.2 Results

As shown in Table 4.3, 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 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.

4.4.5 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 is of high quality, and the NCBI dataset, which is a 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 4.4.3), 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.

4.5 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.

4.5.1 Adaptive Forms

The approach proposed in this chapter 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* [FS98] or *form definition languages* [BBG11] 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) [TB96] or through server-side validation [BBG11]. The dynamic behaviour of a form has also been modeled using a declarative, business process-like notation (DCR - Dynamic Condition Response graph [SLDM18]), 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 [ER04, AR10] 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.

4.5.2 Comparing LACQUER with LAFF

The overall architecture (including the use of the endorser module) of LACQUER has been inspired by the one of LAFF, a recent approach for automated form filling of data entry forms [BLBB22]. In this subsection, we explain the similarities and differences between the two approaches.

4.5.2.1 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 order of filling and (2) with 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 to be used by 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.

4.5.2.2 Differences between LAFF and LACQUER

Table 4.4 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 in LACQUER, textual fields have only two values, i.e., “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 4.3.1).

As for the *model building* phase, LAFF and LACQUER create a different number 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 in terms of the *prediction* phase can be viewed from two perspectives: the type of targets and the endorser module. Concerning

Table 4.4: 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 fields
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

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; LACQUER includes a threshold determination phase to automatically decide the threshold for each target.

4.5.3 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 [MPFN18]), requirement engineering (e.g., to predict the requirement complexity in order to assess the effort needed to develop and test a requirement [SAF22]), implementation (for code auto-completion [PLM15]), quality assurance (e.g., for defect prediction [JBM11, DVB12]), and software maintenance [RJMFSC23].

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 [RJMFSC23]. 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. [RJMFSC23] 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 [DÁDS16, OY14]. For example, in defect prediction, some metrics of defect prediction datasets might be missing for some software modules. To solve this challenge, Del Águila and Del Sagrado [DÁDS16] and Okutan and Yıldız [OY14] 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 receive a huge amount of entries every day.

4.6 Discussion

4.6.1 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 $\geq$ 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 4.4.2, 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.

4.6.2 Practical Implications

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

4.6.2.1 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.

4.6.2.2 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.

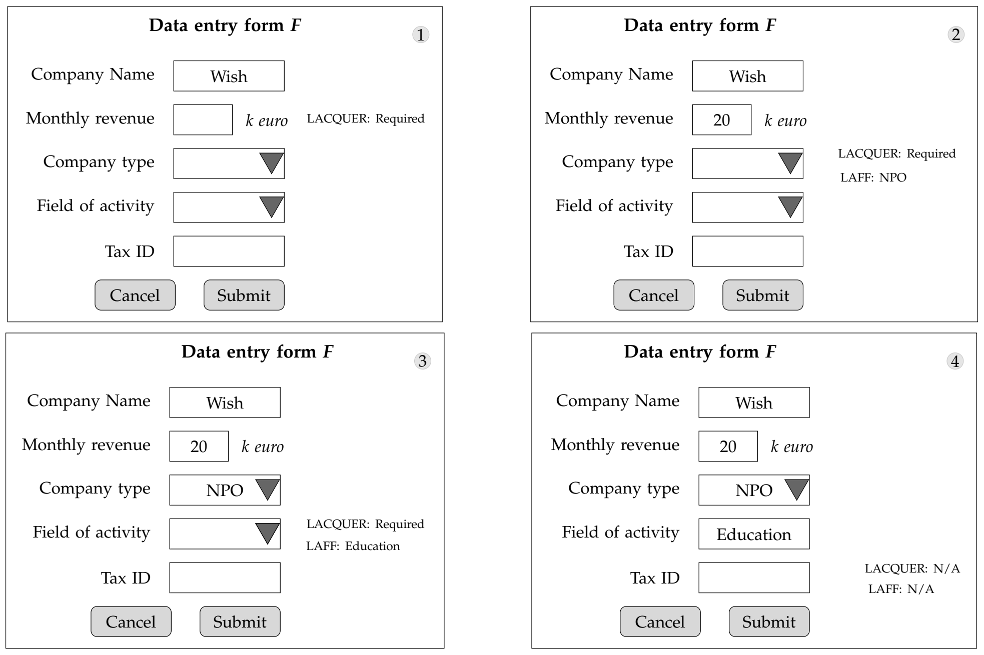
4.6.2.3 Researchers

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.

4.6.3 Combining LACQUER with LAFF

Despite the differences explained in section 4.5, 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 4.7 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, LAFF can already check whether the “monthly income” field is required or not. Since “monthly income” is a numerical field, LAFF cannot



Data entry form *F* 1

Company Name	Wish	
Monthly revenue	*k euro*	LACQUER: Required
Company type		
Field of activity		
Tax ID		

Cancel Submit

Data entry form *F* 2

Company Name	Wish	
Monthly revenue	20 *k euro*	
Company type		LACQUER: Required LAFF: NPO
Field of activity		
Tax ID		

Cancel Submit

Data entry form *F* 3

Company Name	Wish	
Monthly revenue	20 *k euro*	
Company type	NPO	
Field of activity		LACQUER: Required LAFF: Education
Tax ID		

Cancel Submit

Data entry form *F* 4

Company Name	Wish	
Monthly revenue	20 *k euro*	
Company type	NPO	
Field of activity	Education	
Tax ID		LACQUER: N/A LAFF: N/A

Cancel Submit

Figure 4.7: Use case to combine LACQUER and LAFF together during form filling

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, there is no need for LACQUER to relax a completeness requirement and the field is numerical and thus not compatible with LAFF.

4.7 Summary

In this chapter 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.

Chapter 5

LAFF-based Anomaly Detection for Categorical Data

5.1 Overview

In this chapter we propose a LAFF-based Anomaly Detection approach ("LAFF-AD" in short) to effectively detect categorical data anomalies. The basic idea of LAFF-AD is to take advantage of the learning ability of LAFF to perform value inference on suspicious data. The output of such inference is used by LAFF-AD to determine the presence of anomalies in data.

LAFF-AD includes three main phases: LAFF offline prediction, anomaly detection, and threshold determination. Similar to LAFF, LAFF-AD starts by learning Bayesian network models on a clean set of data. Given a set of suspicious data, LAFF-AD runs a variant of LAFF that handles offline prediction (i.e., in contrast to real-time during the data entry process) to predict the value of a categorical field in a suspicious instance. This variant returns a ranked list of possibly correct values for the suspicious instance, the probability of each value in the ranked list, and a flag indicating whether the prediction comes with high confidence. In the next phase, LAFF-AD leverages the output of LAFF to detect data anomaly with a heuristic-based strategy. LAFF-AD analyzes the outputs of LAFF by checking three heuristics. The first one checks if LAFF has enough confidence to make a prediction (i.e., the prediction is endorsed). The second one checks if the suspicious value is ranked to the top in the list predicted by LAFF. The third heuristic checks if the anomaly score of the suspicious value is higher than a certain anomaly score threshold. The anomaly score is computed based on the probability of the suspicious value in the ranked list predicted by LAFF. According to the three heuristic rules, LAFF-AD determines the presence of a data anomaly. In the threshold determination phase, the value of the anomaly score threshold used in the anomaly detection phase is automatically determined for each dataset to minimize the influence of selecting appropriate parameters.

We evaluated LAFF-AD using six datasets with different characteristics. Five of them are labeled datasets widely used to evaluate anomaly detection approaches for categorical data. The last dataset represents a real-world dataset from the biomedical domain, already used to evaluate LAFF [BLBB22]. This dataset is used to further confirm the generalisability of our evaluation since it contains different kinds of anomalies that are synthetically created. The experimental results show that LAFF-AD can accurately detect a high range of data anomalies, with recall values between 0.6 and 1) and a precision value of at least 0.808. The results also show that LAFF-AD is fast enough to be applied to detect data anomalies in practice: LAFF-AD takes at most 7000 s and 735 ms to perform training and prediction, respectively.

To summarize, the main contributions of this chapter are:

- The LAFF-AD approach, which addresses the problem of anomaly detection for categorical data. To the best of our knowledge, LAFF-AD is the first work to repurpose a form filling recommender system to detect data anomalies. LAFF-AD provides effective data anomaly detection results without the need for manual tuning.

- An extensive evaluation assessing the effectiveness and efficiency of LAFF, including a comparison with SOTA approaches. Our evaluation shows that LAFF-AD yields stable results outperforming SOTA algorithms for most datasets.

The rest of the chapter is organized as follows. Section 5.2 provides a motivating example and presents the concept of data anomaly detection. Section 5.3 reviews the state of the art and its limitations. Section 5.4 describes the core phrases of LAFF-AD. Section 5.5 reports on the evaluation of LAFF-AD. Section 5.6 discusses the usefulness of LAFF-AD, taking into account the practical implications of the experimental results. Section 5.7 concludes the chapter.

5.2 Data anomaly detection

5.2.1 Motivating example

Real-world data contains data anomalies that affect data quality. These anomalous data can have severe consequences, especially in critical domains such as finance and health care. Let us assume a dataset for an energy provider contains the following columns "Tariff plan", "Customer segment", "Fixed fees", and "Consumption average" (as shown on the right of Figure 5.1). "Tariff plan" and "Customer segment" are two categorical columns with the following values ("`Standard`", "`Time of use`", "`Renewable energy`") and ("`Residential`", "`Commercial`", "`Non-profit organization`", "`Industrial`"), respectively. The remaining columns "Consumption average" and "Fixed fees" are numerical columns. Based on this information, the energy provider relies on an ML-based bill calculator to decide the "Rate per kWh" according to the needs and the tariff plan of different customers. After a certain period of time, some customers with a standard tariff plan started complaining about extra charges even though their consumption of energy had not increase.

The energy provider team decided to investigate the reason behind this problem. The investigations showed that the main reason for the extra charges is that the bill calculator relied on bad quality data containing some anomalies. The decisions were based on anomalous instances having "Tariff plan" equal to "`Standard`", "Customer segment" equal to "`Industrial`", "Average consumption" equal to "`400`", and "Fixed Fees" equal to "`20`". These instances deviate from the normal data instances since "Tariff plan" equal to "`Standard`" is usually associated with "Customer segment" equal to "`Residential`" and "Average consumption" equal to "`100`". These anomalous instances misled the bill calculator, which used a high "Rate per kWh" equal to "`0.5`" instead of "`0.1`" for residential customers. Indeed, residential customers pay fewer taxes for energy since they use it for personal reasons; on the opposite, industrial customers need to pay more. These anomalies in the data can be introduced in different ways such as during data entry (e.g., typos), data management (e.g., faulty data source), and data integration (when assembling data from different sources) [MBM15].

To solve this problem, the energy provider decides to do an audit on the dataset to check the data quality of different data instances, and to apply anomaly detection in order to detect anomalous instances that affect the data and decision quality.

5.2.2 Problem Definition

In this chapter, we deal with the problem of anomaly detection for categorical data in relational databases. This problem can be informally defined as the problem of deciding whether the value of a target column in a given suspicious instance is anomalous or not.

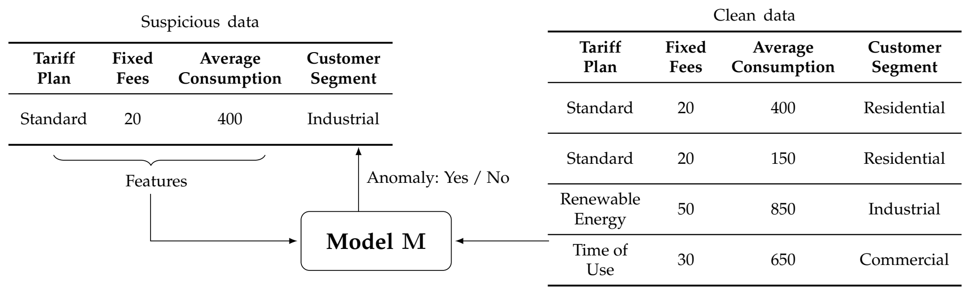
In this work, we target categorical columns since in practice data is often described with categorical attributes [IPM16]. This type of column is subject to anomalous values since, for example, the filling process of categorical attributes is error-prone and time-consuming [BLBB22]. Another reason is that identifying anomalies in such kinds of columns is difficult since it is not easy to devise criteria to separate between anomalous and non-anomalous data [AFPS22].

We define the anomaly detection problem as follows. Let D be a dataset composed of a set of n columns $C = \{c_1, c_2, \ldots, c_n\}$. Let $C^c \subseteq C$ be the set of categorical columns. Each column c_i can take a value from a certain domain V_i. D can be partitioned into datasets CD and SD, representing respectively clean data and suspicious data possibly containing data anomalies; we have $CD \cup SD = D$ and $CD \cap SD = \emptyset$. During the anomaly detection process, the columns are partitioned in two groups, i.e., a set of features C^f and one target column $C^t \in C^c$; we have that $C^f \cup C^t = C$ and $C^f \cap C^t = \emptyset$. Let $v_t^{co} \in V_t$ be the correct value that the target column C^t should have for a given instance, and $v_t^o \in V_t$ be the observed value. In other words, v_t^o is the observed value of the column C^t in the suspicious instance. In our definition, we define an anomaly when the observed value is different from the correct value, i.e., when $v_t^{co} \neq v_t^o$.

The anomaly detection problem can be defined as follows. Given a clean subset of the data CD, a set of features C^f, and a target column C^t, we want to build a model M that can decide if the observed value of the target is anomalous or not (i.e., $v_t^{co} \neq v_t^o$) based on C^f and CD.

5.2.2.1 Application to the running example

Figure 5.1 shows an example illustrating the anomaly detection problem. We have a dataset of an energy provider with four columns, c_1: "Tariff plan", c_2: "Fixed Fees", c_3: "Average consumption", and c_4: "Customer Segment". Among the columns, "Tariff plan" and "Customer Segment" are categorical columns $C^c = \{c_1, c_4\}$. The table on the right-hand side of the figure represents the clean data CD and the small table in the left represents the suspicious data SD. Using the clean data, we want to build a model M to



Tariff Plan	Fixed Fees	Average Consumption	Customer Segment
Standard	20	400	Industrial

Tariff Plan	Fixed Fees	Average Consumption	Customer Segment
Standard	20	400	Residential
Standard	20	150	Residential
Renewable Energy	50	850	Industrial
Time of Use	30	650	Commercial

Figure 5.1: Running example for problem definition

learn relationships between columns c_1 to c_4 in the clean data. This model is then used to check any instance on the suspicious data. Going back to the example, let us assume that we want to check if the value of the "Customer Segment" column in the suspicious test instance is an anomaly or not (i.e., C^t = "Customer Segment"). In this case, as shown in the figure, the rest of the columns are considered as features, i.e., $C^f = \{c_1, c_2, c_3\}$. Our goal is to use the model M to predict whether the observed value of "Customer Segment" (i.e., v_i^o = "Industrial") represents an anomaly or not based on the values of $C^f = \{c_1, c_2, c_3\}$ and the clean data CD.

5.3 State of the art

The approach proposed in this chapter is related to anomaly detection for categorical data. Many research works tried to tackle this problem. The proposed approaches can be classified based on different aspects. In this section we followed the classification proposed by Taha and Hadi [TH19]: frequency-based, Bayesian/conditional frequency-based, density-based, clustering-based, distance-based, information-theoretic, and unsupervised/semi-supervised.

Frequency-based approaches [PCC16, PTAJ16] rely on the frequency of categories to detect data anomalies. Less frequent categories are more likely to be anomalies. A typical frequency-based method is CBRW [PCC16], which relies on two kinds of distributions (i.e., "intra-feature" and "inter-feature" distributions) to detect anomalies. The first one computes the frequency of categories on the target column, while the second one analyzes the distributions of categories in different columns. Based on these two distributions, CBRW computes an outlier score, and returns M instances with the highest outlier score as anomalies.

Bayesian/conditional frequency-based approaches [DS07, RHH11, NK08] follow another definition of anomalies: no matter whether the categories are frequent or not, an infrequent combination of categories is considered as an anomalous instance. These approaches compute a rareness score between categorical values in an instance. Instances with a rareness value less than a certain threshold are considered anomalous.

Density based approaches detect anomalies in subgroups of data (referred as "local area") that share similar characteristics. A typical density-based method is WATCH [LZPQ18]. WATCH detects anomalies in two phases, feature grouping and anomaly detection. First, it regroups related columns (i.e., features) having the same meaning or correlated to each other in the same group. Then, in the second phase, it detects anomalies in these groups by computing an anomaly score for each instance in each feature group. A higher anomaly score indicates a higher probability that an instance is anomalous regarding a feature group. WATCH declares the M instances with the highest anomaly score in each group as anomalies. The algorithm takes the union of anomalies sets in all feature groups to determine the anomalous instances. WATCH determines at most $M \times g$ instances as outliers, where g represents the number of features groups .

Clustering-based approaches [SMA12, SMA13] define anomalous instances as the ones located in a sparse region from other clusters. For example, ROAD [SMA12] determines k clusters on the data using the k-mode algorithm. Then it defines a set of big clusters having a number of instances higher than a certain threshold. To detect anomalies, ROAD computes the distance between the test instance and different clusters. A test instance has a higher chance to be an anomalous instance, if it has a larger distance with the nearest big cluster. ROAD finishes by providing M instances with the highest distances.

Distance-based approaches regard data instances far from the majority of instances as anomalies. For example, ORCA [BS03] defines anomalous instances as the M instances with the highest anomaly score, which is computed as the average Hamming distance between an instance and its k nearest neighbors.

Information theoretic approaches transform the problem of anomaly detection into an optimization problem [HDXH06, HDX05]. These approaches use information entropy to detect anomalous instances. One instance is considered to be an anomaly if the entropy of the dataset exhibits a large decrease after removing the instance. Specifically, these approaches first compute the entropy of the original dataset. Then, for each instance, they remove it in the dataset, compute a new entropy value of the dataset, and finally determine the difference between the original entropy value and the one obtained after removing the instance. The k instances with the highest difference in entropy value are selected, and returned as anomalous instances.

Unsupervised/semi-supervised approaches work as follows. Unsupervised anomaly detection approaches are used when there is no information about the anomaly labels (i.e., anomalous or non-anomalous) of data instances. The baselines iForest, LOF, and EMAC are traditional unsupervised anomaly detection approaches. iForest [LTZ12] relies on an ensemble of decision trees to detect data anomalies where it decides instances with shorter average paths as anomalous. LOF [BKNS00] compares the density of one instance with its k nearest neighbors. Then it classifies the instances with lower density values when compared to neighbors as anomalies. EMAC-SCAN [XWWW19] is another unsupervised method, that takes advantage of embedding-based approaches to capture the relationship between categorical features and use them for anomaly detection.

Semi-supervised approaches take advantage of existing non-anomalous instances. In the training phase, they create a novelty model over data instances. Any test instance that deviates from normal data is considered anomalous. Our baseline OCSVM [CSFS02] uses the training data to find a hyperplane separating between anomalous and normal data; this hyperplane is used during the anomaly detection phase where all instances with a high distance to the hyperplane are considered as anomalous. Frac [NBS12] uses non-anomalous instances to build an ensemble of classification models; during the test phase, Frac uses the predictions of previously trained models to determine anomalies. A test instance is considered as anomalous if there is a disagreement among the outputs of the different models. However, Frac is not designed for categorical data. To detect categorical anomalies with Frac, it should rely on any algorithm that can deal with categorical variables.

5.3.0.1 Limitations

Our preliminary experiments on commonly-used anomaly detection datasets show that the effectiveness of the SOTA approaches is unstable. For example, the precision of both iForest [LTZ12] and OCSVM [CSFS02] can vary between 0.03 and 0.9 depending on the datasets. Moreover, existing approaches are highly sensitive to the configuration parameters [YS20]. These algorithms include different parameters (e.g., OCSVM has two parameters) which need to be carefully chosen. In this work, we address these challenges and improve the anomaly detection precision. To detect data anomalies effectively, an intuitive solution is to infer the correct value in a categorical column; then, data anomaly can be detected by comparing the inferred correct value with the observed one. Another advantage is that, using such inferred value, a data anomaly can also be easily fixed.

5.3.0.2 Data Anomaly and Form Filling

Since one of the main sources of data anomaly is the wrong values entered by users during form filling [SZ03], in the literature many form filling approaches [BLBB22, MROE$^+$19] have been proposed to accurately predict the correct value to be filled in a data entry form. In this work, we repurpose an automated form-filling approach to detect data anomalies. To do so, we need first to adapt these approaches to perform predictions offline instead of online (during the form-filling process). Moreover, form-filling approaches typically return to the user a ranked list of items. In order to detect anomalies, we need to fully take advantage of all this information. The main challenge is how to fully leverage the characteristics and outputs of form-filling approaches to effectively perform data anomaly detection.

5.4 Approach

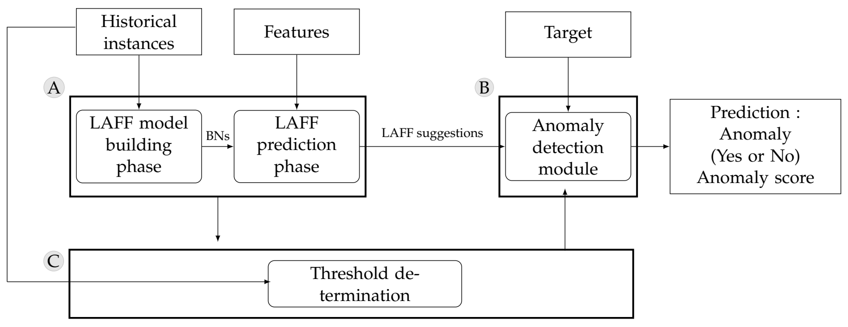
In this section, we show how to repurpose an automated form filling approach LAFF for performing anomaly detection of categorical data; we call the resulting approach LAFF-AD (LAFF-based Anomaly Detection).

As shown in Figure 5.2, LAFF-AD includes three phases: LAFF offline prediction, anomaly detection, and threshold determination. LAFF-AD starts by running a variant of LAFF on historical instances to train BN models. Then LAFF is used to predict the values of the target on suspicious data. In the second phase, based on LAFF's prediction, LAFF-AD uses a heuristic anomaly detection strategy to decide if there is an anomaly or not for a given test instance. LAFF-AD uses a threshold determination phase to automatically decide the values of its parameters.

5.4.1 LAFF offline prediction

LAFF is mainly designed to predict a ranked list of values for a categorical field during the form-filling process. LAFF-AD repurposes LAFF and takes advantage of it to detect data anomalies in categorical columns. The main reason behind repurposing LAFF to detect data anomalies is the high ability it has shown to correctly provide suggestions during form filling [BLBB22].

In the context of anomaly detection, LAFF needs to perform predictions offline on suspicious data that may contain anomalies. This means that LAFF-



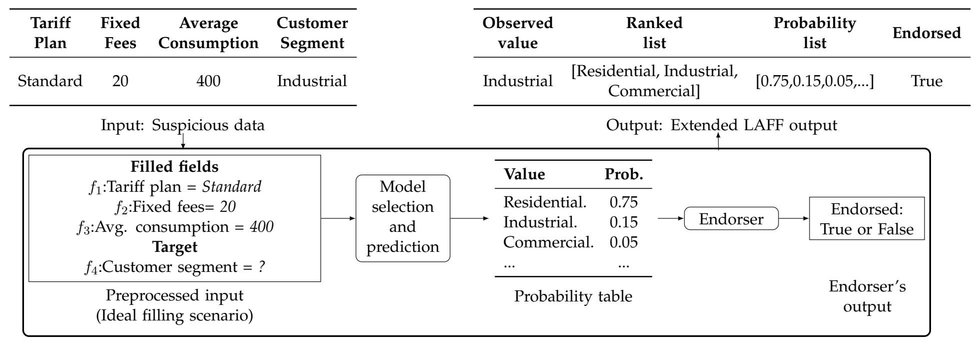
A LAFF Prediction Phase, B Anomaly Detection Phase, and C Threshold Determination Phase

Figure 5.2: Main Steps of LAFF-AD

AD runs an variant of LAFF that handles offline prediction. This variant represents a special case during the form filling process where all fields are filled except the target field. The model building phase is the same as in the original definition of LAFF. LAFF starts by preprocessing the historical instances and then creates different BN models (global and local models).

During the prediction phase, LAFF considers each instance in the suspicious data as data entered by a user. Since we aim to detect anomalies for a given categorical column (i.e., the target column) based on the values of other columns (i.e., the features), we consider the features as filled fields and use LAFF to predict the value of the target column on each suspicious instance. As we mentioned before, LAFF first tries to select one model to predict. This model is selected based on the feature values of the suspicious data. LAFF then makes predictions and returns a ranked list of values based on their probability distribution. Thanks to its endorser module, LAFF has the ability to avoid inaccurate suggestions: it can label a suggestion as not endorsed if it has no sufficient confidence in the suggestion.

Since LAFF-AD repurposes LAFF to detect data anomalies, the output of LAFF needs to be adapted to our context. Our variant of LAFF transforms the output of LAFF into a table representing a summary of the prediction result. This table contains the following information: “Observed value” representing the original value in the suspicious data, “Ranked list” representing the candidate values in the order suggested by LAFF, “Probability list” containing the probability of each value in the ranked list, and a Boolean column “Endorsed” that records whether the prediction of the current test instance is endorsed.



Tariff Plan	Fixed Fees	Average Consumption	Customer Segment
Standard	20	400	Industrial

Observed value	Ranked list	Probability list	Endorsed
Industrial	[Residential, Industrial, Commercial]	[0.75,0.15,0.05,...]	True

Value	Prob.
Residential.	0.75
Industrial.	0.15
Commercial.	0.05
...	...

Input: Suspicious data
Output: Extended LAFF output
Filled fields
f_1:Tariff plan = *Standard*
f_2:Fixed fees= *20*
f_3:Avg. consumption = *400*
Target
f_4:Customer segment = *?*
Preprocessed input (Ideal filling scenario)
Model selection and prediction
Probability table
Endorser
Endorsed: True or False
Endorser's output

Figure 5.3: LAFF prediction phase

5.4.1.1 Application to the running example

Let us consider a suspicious test instance identified by domain experts. As shown on the left-hand side of Figure 5.3, this table contains the following values: "Tariff plan"= "*Standard*", "Fixed fees" = *20*, "Average consumption"= *400*, and "Customer segment" = "*Industrial*". This instance is considered to be anomalous because it deviates from the normal data instances, in which the value "*Standard*" for "Tariff plan" is usually associated with the value "*Residential*" for "Customer segment" and the value *100* for "Average consumption" (see Section 5.2.1). Let us assume that we want to check whether the "Customer segment" value is correct. In this case, the "Customer segment" column is the target and the remaining columns are features. LAFF preprocesses these values and uses them as input for model selection and prediction. Let us assume that LAFF predicts the following values "*Residential*", "*Industrial*", "*Commercial*" with the probability values *0.75, 0.15, 0.05*, respectively; the predicted list is checked by the endorser module of LAFF. Let us assume that the endorser threshold is equal to 0.8; in this case, the prediction should be endorsed since the sum of the top n values (0.75+0.15+0.05 = 0.95) is higher than the endorser threshold. The detailed output of this variant of LAFF is shown in the top-right part of the figure, in the form of a table. The original value in the suspicious data ("*Industrial*") is shown in column "Observed value"; the candidate values (in the order suggested by LAFF) and the corresponding probability values are shown in the second and third columns; the last column indicates the output of the endorser module ("True").

Algorithm 6: Anomaly detection

```
Input: Triple of ranked values, probability list, endorser decision $\langle rv, pl, endorsed \rangle$
       Target column $C^t$
       The value of $C^t$ in the suspicious data: $v_t^o$
       Threshold $\theta_t$
Output: A flag $checkERR_t$, representing the decision to label the suscpicious value
        $v_t^o$ as an anomaly
1  Boolean $checkEndorsed_t \leftarrow isEndorsed(endorsed)$;
2  topRankedValues $\langle rv_{top}, pl_{top} \rangle \leftarrow getTopNpRanked(rv, pl)$;
3  Boolean $checkNotTop_t \leftarrow isNotInTopNp(rv_{top}, v_t^o)$;
4  Float anomalyScore $\leftarrow 1 - getProb(v_t^o, pl)$;
5  Boolean $checkProb_t \leftarrow (anomalyScore > \theta_t)$;
6  $checkERR_t \leftarrow False$;
7  if $checkEndorsed_t$ then
8      if $checkNotTop_t$ and $checkProb_t$ then
9          $checkERR_t \leftarrow True$
10     end
11 else
12     $checkERR_t \leftarrow Not\ Conclusive$
13 end
14 return $checkERR_t$;
```

5.4.2 Anomaly detection phase

This phase is the main phase to detect data anomalies. It assumes that LAFF made a prediction over an instance from the suspicious data. LAFF-AD takes the output of LAFF as determined in the previous phase and uses it to detect anomalies based on a heuristic. The main steps of the anomaly detection algorithm are shown in Algorithm 6. The inputs of the algorithm are the output of LAFF (consisting of the ranked values, the probability list, and the endorser decision), the target column C^t, its original value in the suspicious data v_t^o, and the anomaly detection threshold θ_t. This threshold is used to decide about the existence of data anomalies and its value is automatically determined (see Section 5.4.3).

Based on the output of LAFF, LAFF-AD collects three Boolean flags needed for anomaly detection: $checkEndorsed_t$, $checkNotTop_t$, and $checkProb_t$. First, LAFF-AD checks if the LAFF prediction was endorsed, and saves the value to the Boolean flag $checkEndorsed_t$ (line 1). In order to assign a value to the $checkNotTop_t$ flag, LAFF-AD collects the list of the top-n ranked values by LAFF (line 2). Then, it checks if the original value in the suspicious data v_t^o is not present in the top-ranked list (line 3). If so, the value of $checkNotTop_t$ is set to *True*. After that, LAFF-AD computes an anomaly score based on the probability of the observed value in the suspicious data v_t^o in LAFF's prediction (line 4) and saves the value in the variable $anomalyScore$. LAFF-AD checks if the value of the $anomalyScore$ is higher than the anomaly detection threshold θ_t, and saves the value in the Boolean flag $checkProb_t$ (line 5).

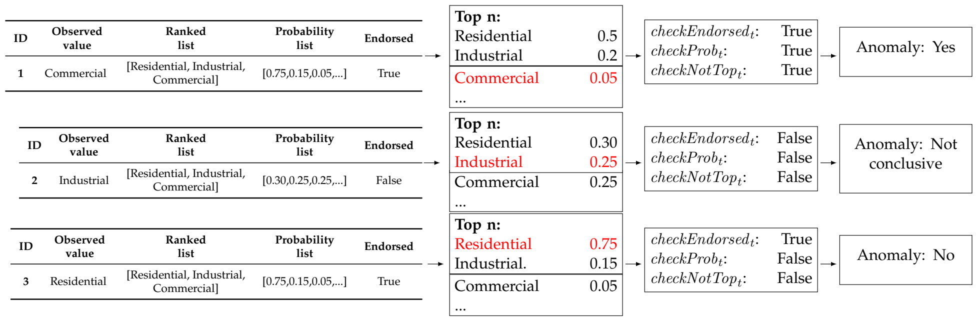
After obtaining these three Boolean flags, LAFF-AD determines the existence of an anomaly based on the following three conditions. The first condition checks if LAFF's prediction is endorsed or not. The second condition checks whether the observed value of the target in the suspicious data is in the "top-n" values predicted by LAFF. The third condition checks if the anomaly score (i.e., $1 - getProb(v_t^o, pl)$) is higher than the anomaly detection threshold, where $getProb(v_t^o, pl)$ represents the probability of the observed value of the target in the ranked list. If the first condition is satisfied, it means that LAFF has enough confidence to predict and thus LAFF-AD can confidently detect anomalies. Regarding the second condition, any observed target value in the suspicious data that does not exist in the ranked list predicted by LAFF represents a potential anomaly, since LAFF usually predicts correct values at the top of the ranked list. As for the third condition, if the anomaly score ($1 - getProb(v_t^o, pl)$) is higher than the anomaly score threshold, this means that the observed value of the target represents a potential anomaly, since the probability of the observed value to be the correct one is very low.

In Algorithm 6, LAFF-AD first checks if the LAFF prediction is endorsed (line 7). If the value of this flag is *True*, LAFF-AD checks the values of flags $checkProb_t$ and $checkNotTop_t$. If both $checkProb_t$ and $checkNotTop_t$ evaluate to *True*, LAFF-AD decides that there is an anomaly and sets the flag $checkERR$ to *True* (line 10). If the LAFF prediction is not endorsed, this means that we do not have enough information to make a prediction. We assign the value "Not conclusive" to the decision flag $checkERR$ (line 12). The algorithm ends with returning the value of the $checkERR$ flag.

5.4.2.1 Application to the running example

Figure 5.4 shows three instances that need to be checked by LAFF-AD. For each instance, we have the output from the LAFF offline prediction step (see section 5.4.1), which is the input for anomaly detection.

For the first suspicious instance in Figure 5.4, LAFF provides an endorsed prediction (*checkEndorsed*$_t$= *True*). In this case, LAFF-AD therefore has enough confidence to make a decision on the existence of an anomaly. LAFF predicts *Residential, Industrial, and Commercial* with the following probability list *0.5, 0.2, and 0.05*, respectively. If we assume that the value of the anomaly



ID	Observed value	Ranked list	Probability list	Endorsed
1	Commercial	[Residential, Industrial, Commercial]	[0.75,0.15,0.05,...]	True

ID	Observed value	Ranked list	Probability list	Endorsed
2	Industrial	[Residential, Industrial, Commercial]	[0.30,0.25,0.25,...]	False

ID	Observed value	Ranked list	Probability list	Endorsed
3	Residential	[Residential, Industrial, Commercial]	[0.75,0.15,0.05,...]	True

Figure 5.4: Examples of LAFF-AD's prediction

threshold for the "Customer segment" column is equal to 0.9, LAFF-AD then predicts the existence of an anomaly since the $checkProb_t$ flag is evaluated to *True* ((1-0.05)$\geq$ 0.9) and the observed value *Commercial* is not in the top-n values predicted by LAFF (i.e., $checkNotTop_p$= *True*).

The second case illustrates the scenario when LAFF-AD does not have the confidence to determine the existence of an anomaly. Specifically, even though the observed value in the suspicious instance is predicted in the top-n ($checkNotTop_t$=*False*) and the anomaly score is less than the threshold ($checkProb_t$= *False*), LAFF-AD decides to set the value of the $checkERR$ to *Not conclusive* since the prediction of LAFF is not endorsed.

The last case shows the scenario when a suspicious instance should not be treated as an anomaly. In fact, as shown in the figure, the prediction is endorsed by LAFF ($checkEndorsed_t$= *True*) and the observed value of the target is predicted at the top of the ranked list (i.e., $checkNotTop_t$=*False*). Concerning the flag $checkProb_t$, as shown in the figure, the probability of the observed value is very high (equal to 0.75) which leads to a low anomaly score (i.e., $checkProb_t$=*False*). Since both of the flags evaluate to *False*, LAFF-AD can confidently decide that this instance is normal.

5.4.3 Threshold tuning phase

In this phase, we aim to automatically determine the value of the anomaly score threshold for each target, in each dataset. This value is used in the anomaly detection algorithm and plays a determinant role. This step assumes that LAFF was run on a set of historical instances for tuning and that the LAFF predictions are available.

The basic idea is that, for a given dataset and target column, we try to detect anomalies in each instance of the tuning data set by varying the anomaly

Algorithm 7: Threshold determination

```
Input: Set of pre-processed historical instances $I^H(t)_{tune}$ for tuning
       LAFF's predictions on historical instance for tuning: list of triples of ranked
       values, pl , endorser decision $\langle rv, pl, endorsed \rangle_{tune}$
Output: Dictionary of thresholds $\theta$
1  $\theta$ ←empty dict;
2  List of targets $targets \leftarrow getTargets(I^H(t)_{tune})$;
3  foreach target $t_i \in targets$ do
4      $temp_{th}$ ←empty dictionary;
5      for $n$= 0 to 1 (step 0.05) do
6          $predictedAnomalyAll =$
             $predictAnomalyAllInstances(I^H(t)_{tune_i}, \langle rv, pl, endorsed \rangle_{tune}, n)$;
7          $score = evaluate(I^H(t)_{tune_i}, predictAnomalyAll)$;
8          $temp_{th}[n] = score$;
9      end
10     $\theta[i] = getBestScore(temp_{th})$;
11 end
12 return $\theta$;
```

detection threshold θ_i. Then, for each threshold, we measure the prediction accuracy on the tuning set instances. The threshold on which LAFF-AD yields the highest accuracy is set as the target threshold.

The main steps of our threshold determination phase are shown in Algorithm 7. First, as inputs, the algorithm takes the set of preprocessed historical instances for tuning $I^H(t)_{tune}$ and LAFF predictions on these historical instances $\langle rv, pl, endorsed \rangle_{tune}$.

For each target t_i in the list of targets extracted from $I^H(t)_{tune}$ (line 2), based on LAFF predictions, we check the different tuning instances and analyze the existence of anomalies using the approach explained in section 5.4.2 (line 6). For the purpose of anomaly detection, we try different thresholds, ranging from 0 to 1 with steps equal to 0.05. For each threshold value, we compare predicted anomalies with actual anomalies of the target t_i in each input instance of $I^H(t)_{tune_i}$ to calculate prediction accuracy (line 7). LAFF-AD selects the value of θ_i that leads to the highest prediction accuracy value for a target t_i in $I^H(t)_{tune_i}$ as the value of its threshold (line 10). The algorithm ends by returning a dictionary containing the thresholds of all targets.

5.5 Evaluation

In this section, we report on the evaluation of our anomaly detection approach. We focus on two aspects, effectiveness and efficiency, which we

Table 5.1: Information of the Datasets

Dataset	# of columns	# of instances	# of categorical columns	Range of candidate values
NCBI-E	25	74105	5	3–84
U2R	7	60821	7	2–23
Probe	7	64759	7	2–47
CelebA	40	202599	40	2
Covertype	45	581012	45	2
Census	34	299285	34	2–47

compare with state-of-the-art approaches. Efficiency is defined in terms of training and prediction time, to understand the suitability of an approach for practical applications. More specifically, we evaluated LAFF-AD by answering the following research questions (RQs):

RQ1 *Can LAFF-AD accurately detect data anomalies on categorical columns and how does it compare with existing anomaly detection approaches?*

RQ2 *Is the performance of LAFF-AD, in terms of training and prediction time, suitable for practical applications and how does it compare with existing approaches?*

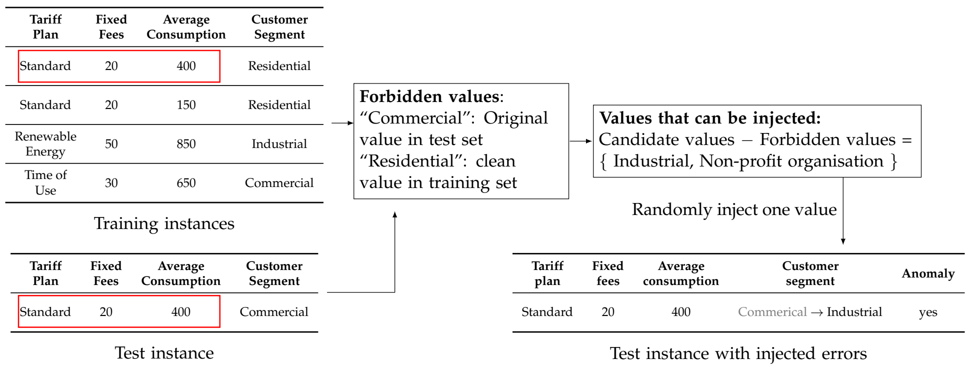
5.5.1 Dataset and Settings

5.5.1.1 Datasets

Table 5.1 shows an overview of the datasets used in our evaluation, including the total number of columns (# of columns), the number of instances (# of instances), the number of categorical columns (# of categorical columns), and the range of the number of possible values across these columns (Range of candidate values).

The first dataset is NCBI-E, which is a variation of the public dataset NCBI from the biomedical domain. The original NCBI dataset contains data for different types of biological samples from multiple species [BCG+12]. The main reason to use this dataset is that it was used to evaluate LAFF [BLBB22]. Similar to LAFF, we considered a sub-sample of the data related to the species "Homo Sapiens". In our evaluation, we created NCBI-E by removing the column *ethnicity* from the NCBI dataset, since as shown in the existing study [BLBB22] only 15.6% of instances are non-empty in this column, which leads to incorrect suggestions from LAFF.

The remaining 12 datasets are from the publicly-available benchmark that is commonly used to evaluate anomaly detection approaches for categorical columns [PCC21]. Based on guidelines provided by Belgacem et al. [BLBB22],



Tariff Plan	Fixed Fees	Average Consumption	Customer Segment
Standard	20	400	Residential
Standard	20	150	Residential
Renewable Energy	50	850	Industrial
Time of Use	30	650	Commercial

Training instances

Tariff Plan	Fixed Fees	Average Consumption	Customer Segment
Standard	20	400	Commercial

Test instance

Forbidden values:
"Commercial": Original value in test set
"Residential": clean value in training set

Values that can be injected:
Candidate values − Forbidden values = { Industrial, Non-profit organisation }

Tariff plan	Fixed fees	Average consumption	Customer segment	Anomaly
Standard	20	400	Commerical → Industrial	yes

Test instance with injected errors

Figure 5.5: Running example of error injection

only the datasets with more than 56 000 instances available for training were selected for our evaluation, since LAFF achieves accurate suggestions only for these datasets: U2R, Prob, CelebA, Covertype, and Census. We therefore excluded the remaining datasets: Bank (41188), AID (4279), W7A (49749), CMC (1473), APAS (12695), Chess (28056), AD (3279), Solar (1066), and R10 (12897).

As shown in Table 5.1, the number of instances in these datasets varies from 60 821 (U2R) to 581 012 (Covertype). These datasets feature at least 7 columns, with more than 5 being categorical. The number of candidate values for these categorical columns varies significantly. For example, in the NCBI-E dataset, categorical columns feature between 3 and 84 candidate values, whereas the Covertype dataset contains only binary categorical columns.

5.5.1.2 Dataset preparation

For the NCBI-E dataset, we considered all the categorical columns as possible targets. It is not mainly designed to evaluate anomaly detection algorithms since anomalous values are not labeled.

To solve this issue, we adopted an anomaly injection strategy which has been used to evaluate anomaly detection algorithms [AA17].

Following the methodology used by Das and Schneider [DS07], we injected synthetic anomalies in the NCBI-E dataset by randomly flipping the values of a target column. Specifically, we partitioned the NCBI-E dataset into three subsets of 80%, 10%, and 10% of instances, used for training, testing, and tuning, respectively. For each target, we created a separate test set where we kept the values in the feature columns the same and injected errors in the target column. We randomly selected a value that is different from the original value in the target column from candidate values. We then replaced the original value with the selected value to inject anomalies, and labeled the instance as anomalous.

For the other datasets, as mentioned by Pang et al. [PCC21], each dataset has one target column that contains an anomalous value. Similar to NCBI-E, we split the datasets into three subsets containing 80%, 10%, and 10% of instances used respectively for training, testing, and tuning.

As suggested in different studies [IPM16, PCC16, SMA13], there is usually a small ratio of anomalous instances in real-world data. For example, the ratio of anomalous instances used by Suri et al. [SMA13] ranged from 10% to 16%. We followed the methodology used by Ienco et al. [IPM16], for all these datasets, the ratio of anomalous instances is set to 10% of instances the data. The anomalous instances are either randomly selected (for the benchmark datasets) or injected (for the NCBI-E dataset).

5.5.1.3 Dataset preparation Example of Application

Figure 5.5 show an example of our error injection process. Let us consider the two tables on the left of the figure as the training (top left) and testing (bottom left) sets. Following our running example, the columns “Tariff plan”, “Fixed fees”, and “Average consumption” represent the features and the column “Customer segment” represents the target (where we want to inject errors).

Given the training and test instances, we need first to determine the set of forbidden values that should not be injected in the target column of the current test instance. This set should obviously contain the values *Commercial*, representing the original value of the test instance. But since we assume that the training set is clean, if there is any training instance with the same feature column values, the value of the target column in this instance must also be part of the forbidden values. For example, based on Figure 5.5, the value *Residential* should also not be injected to avoid having instances that are considered clean during training but anomalous during testing. After determining forbidden values, the remaining candidate values for the target (i.e., “Industrial” and “Non-profit organisation” in our example) can be injected into the test instance. In our example, we randomly select the value “Industrial” to be injected. After injection, this instance is considered “Anomalous” and we label it as such (i.e., filling ‘yes’ in the column Anomaly in the table on the right).

5.5.1.4 Implementation and Setting

LAFF-AD is implemented as a Python program. In order to run LAFF on different datasets, we used its default configurations mentioned in [BLBB22]. We performed experiments with a computer running macOS 10.15.5 with a 2.30 GHz Intel Core i9 processor with 32 GB memory.

5.5.2 Effectiveness (RQ1)

To answer RQ1, we analyzed anomaly detection with LAFF-AD for each of the targets in different datasets. We compared LAFF-AD with iForest (Isolation Forest) [LTZ12], LOF (Local Outlier Factor) [BKNS00], OCSVM (One Class SVM) [CSFS02], and EMAC-SCAN (Embedding-based coMplex vAlue Coupling learning framework) [XWWW19]. These approaches are commonly used as baselines to evaluate categorical data anomaly detection approaches [IPM16, PTAJ16]. Moreover, there are publicly available replication packages including their implementations.

LOF uses the distance between instances to detect data anomalies. Given a data instance, LOF compares the distance between the instance and its nearest neighbors to assess density, which measures how closely packed the data instances among those neighbors. A data instance is considered an anomaly if it has a lower density value compared to its neighbors. In other words, anomalous data instances are relatively far from local groups.

OCSVM uses the training instances to iteratively find a hyperplane that separates normal instances from anomalies. In order to detect anomalous instances, OCSVM computes the distance between new instances and the hyperplane. Instances with high distances are marked as anomalous.

iForest builds an ensemble of decision trees over a given dataset. These decision trees are used to detect anomalous instances based on the number of splits needed to separate data instances. The intuition behind this algorithm is that anomalous data instances can be easily separated from normal instances. Based on this intuition, iForest classifies instances with a smaller average number of splits as anomalous.

EMAC-SCAN starts by embedding values of categorical features into continuous vectors by employing a skip-gram architecture (i.e., node2vec). These vectors represents the relationships between different categorical

Table 5.2: Main metrics used in the area of anomaly detection for categorical fields

Metric	Description	Cited papers
Precision	The fraction of correctly detected anomalies among all the predicted anomalies [RHH11].	5
Recall	The fraction of correctly detected anomalies among all the anomalous instances [RHH11].	7
Accuracy	The fraction of correctly predicted anomalies among the total number of predictions. [RHH11]	3
Number of detected anomalies	The number of correctly detected anomalies.	2
F_1*-score*	The harmonic mean of precision and recall [APGG14].	1
ROC curve	ROC plots the true positive rate and false positive rate of a classification model [HCT17].	4
AUC	The ability of a binary classifier to distinguish between classes; it is used as a summary of the ROC curve [Bra97].	9

values. Then, the algorithm learns a coupling function that assigns an anomaly score to each vector. The algorithm considers instances with an error score higher than a certain threshold as anomalies.

5.5.2.1 Choosing effectiveness metrics

In order to select the evaluation metric, we checked around 25 papers included in a recent survey [TH19] on anomaly detection algorithms for categorical data. We also checked other recent papers citing these papers.

Table 5.2 presents a summary of our literature review. This table contains three columns, showing the metrics, their description, and the number of papers using each metric. As shown in the table, seven metrics have been commonly used. Precision, Recall, AUC (Area under Curve), and ROC (Receiver Operator Characteristic) curve are the most used ones. AUC is based on the ROC curve, which is usually used to evaluate the performance of a classification model under different thresholds. We do not select AUC and ROC since LAFF-AD automatically decides the error threshold during the tuning phase. In order to evaluate our approach, we then simply rely on precision and recall.

5.5.2.2 Methodology

We assessed the accuracy of different algorithms using Precision (Prec) and Recall (Rec), which are computed from the confusion matrix summarizing the classification outputs. The confusion matrix includes True Positive (TP), False Positive (FP), True Negative (TN), and False Negative (FN). In our context, TP are correctly classified as an anomaly, FP are wrongly classified as an anomaly, TN are correctly predicted as not anomalous, and FN are misclassified as not anomalous. Based on the confusion matrix, we have $Prec = \frac{TP}{TP+FP}$ and $Rec = \frac{TP}{TP+FN}$.

Precision is the ratio of correctly detected anomalies over all the values classified as anomalies. Recall is the ratio of correctly predicted anomalies over all actual anomalies. High values of precision and recall imply that an algorithm can correctly detect most anomalies. In the scenario of anomaly detection, the main goal is to successfully detect all the anomalies in a dataset, while avoiding predicting non-anomalous values as anomalies. In other words, our approach needs to have both high precision and recall values.

Regarding the four baselines, OCSVM, LOF and iForest are used to detect anomalies in numerical data [PTAJ16]. To be able to run these algorithms on our categorical data, it is necessary to convert categorical columns into numerical ones. One common conversion method used in categorical anomaly detection is *1-of-l* [HFH$^+$09]. This method converts a categorical column with *l* candidate values to *l* binary columns representing each categorical value. Such a column can take the values "1" or "0", representing respectively if the categorical value associated with the column is selected or not. Further, these algorithms have different parameters to set, which may affect their effectiveness. Since the values of these parameters may vary from one dataset to another, in order to ensure a fair comparison, we explored the settings used for these parameters in related works [XWC$^+$18, LTZ12, CSFS02]. For each algorithm, if there is only one parameter value used in different papers, we consider it as default value for this parameter. If a parameter was set with different values in related works, we tune this parameter using grid search. The tuning follows this strategy: for each parameter we explore a tuning range from the minimum to the maximum values reported in the literature in steps of 10 or 0.1; for each algorithm we then use the parameter values with the highest accuracy for a given dataset.

Table 5.3: Effectiveness for Anomaly detection

	Alg.	Accuracy		Train	Predict (ms)	
		Prec	Rec	(s)	avg	min–max
NCBI-E	OCSVM	0.202	0.707	21	3	3–3
	LOF	0.164	0.925	61177	33	27–57
	iForest	0	0	264	36	30–64
	EMAC-SCAN	0.213	0.828	0	1	0.767–2
	LAFF-AD	0.808	0.970	1522	14.81	6–37
U2R	OCSVM	0.016	1	10585	5	4–9
	LOF	0.384	0.333	24813	0.6	8–13
	iForest	0.169	0.733	43	23	23-23
	EMAC-SCAN	0.562	0.600	0	0.74	0.71–0.75
	LAFF-AD	1	1	535	6	5–15
Probe	OCSVM	0.183	1	2136	3	3–3
	LOF	0.770	0.241	14330	7	7–7
	iForest	0.940	0.892	52	27	23–48
	EMAC-SCAN	0.864	0.851	0	0.864	0.860–0.868
	LAFF-AD	0.981	0.266	984	8	5–37
CelebA	OCSVM	0.052	0.535	709	2	2–2
	LOF	0.005	0.005	10359	10	9–14
	iForest	0.134	0.153	204	18	17–22
	EMAC-SCAN	0.112	0.295	0	21	18-25
	LAFF-AD	1	1	6187	479	443–712
Covertype	OCSVM	0.003	1	3085	2	2–2
	LOF	0.107	0.085	9630	12	11–16
	iForest	0.124	0.554	643	21.172	20–25
	EMAC-SCAN	0.036	0.973	0	95	93– 97
	LAFF-AD	1	0.570	4450	476	425–735
Census	OCSVM	0.062	0.211	10916	5	5–5
	LOF	0.008	0.002	18208	74	63–105
	iForest	0.047	0.015	1504	30	24–38
	EMAC-SCAN	0.054	0.649	0	74	74 – 80
	LAFF-AD	0.970	0.928	6710	186	27-330

5.5.2.3 Results

Table 5.3 shows the results of the various algorithms on the different datasets used in our evaluation. Column *Alg* indicates the algorithm, while columns *Prec* and *Rec* indicate the precision and recall values, respectively.

Starting with the results on the NCBI-E dataset where anomalies are randomly injected, LAFF-AD significantly outperforms baselines in terms of Prec and Rec, ranging from +59 pp to +64 pp for Prec and +4.5 pp to +26.3 pp for Rec. When we compare the recall value of LAFF-AD and LOF, both approaches have similar recall value but LAFF-AD performs much better in terms of Prec (0.808 vs 0.164).

Looking at the results on the benchmark datasets, LAFF-AD substantially outperforms the baselines for the U2R, CelebA, Census, and Covertype datasets where the value of both Prec and Rec are almost equal to 1. More in detail, LAFF-AD outperforms all the baselines on these datasets in terms of precision by at least 43.8 pp; in terms of recall, LAFF-AD outperforms the other baselines by at least 26 pp, except for the Covertype dataset where the recall of EMAC-SCAN is higher than LAFF-AD by around 40 pp. In this last case, we remark that, even though the recall value of EMAC-SCAN is very high compared to LAFF-AD, the precision value is very low (Prec=0.036). This means EMAC-SCAN always predicts instances as anomalous. On the contrary, LAFF-AD can accurately detect almost 60% of the anomalous instances with a precision equal to 1.

Looking at the results of the Probe dataset, we can notice that both baselines iForest and EMAC-SCAN outperform LAFF-AD in terms of Rec by almost 60 pp. Also, these baselines provide accurate suggestions with Prec equal to 0.940 and 0.864 for iForest and EMAC-SCAN, respectively. We analyze the reasons of the weak results of LAFF-AD on the Probe dataset in § 5.5.2.4 below.

Comparing the results of the baselines between (Probe, U2R) and (Census, Covertype and CelebA), we notice that the results of the baselines are very low for Census, Covertype and CelebA, especially in terms of Prec. One possible reason can be the values of the parameters used in these baselines; however, we have tried to use tuning to solve this problem. Besides, we believe that the high number of columns in the dataset (34 and 45 columns) may be the reason of the low precision [KSZ08]. High-dimensional datasets (i.e., with high number of columns) are challenging ML algorithms [AHWM20]. High dimensionality can make it difficult for machine learning algorithms to learn information from observed data, a problem referred to as the curse of dimensionality [Rus10]. For example, the Census dataset is composed of 34 categorical columns where the range of number of candidate values varies from 2 to 47. As we mentioned before, we need to transform these columns to binary columns in order to run our baselines. The number of columns in the transformed Census dataset is equal to the sum of the number of candidate values of each categorical column in the dataset; this value may affect the ability of our baselines.

The results in terms of Precision and Recall values achieved by LAFF-AD show that it can help correctly ($prec \geq 0.808$) detecting a high ratio of data anomalies in different datasets with a recall ranging from 0.570 to 1.

Table 5.4: Maximum Carmér's V for different datasets

Dataset	Max Carmér's V	Column
U2R	0.664	Service
Probe	0.848	Flag, Service
CelebA	0.223	Att13
Covertype	0.275	Dim-17
Census	0.438	Att4

5.5.2.4 Error analysis

As presented in Table 5.3, the recall value of LAFF-AD for the *Probe* dataset is low compared to the results of LAFF-AD on other benchmark datasets. In order to understand the reason, we checked the prediction details of LAFF-AD in *Probe* and found that all the missed anomalies predicted by LAFF-AD have an anomaly score (as defined in § 5.4.2) of 0. This means that the form filling tool LAFF predicts the anomalous values in the top first position of the ranked list with a prediction confidence of 100%. Here the anomalous values represent values or dependencies that seldom occur in the data. LAFF-AD cannot detect the anomaly because the anomalous values are predicted in the top of the ranked list (leading to $checkNotTop = False$ when executing Algorithm 6) and with low anomaly score (leading to $checkProb = False$).

We further analyze the reason behind the high confidence of the prediction of the anomalous values. To do so, we performed the Chi-square test [Fie79] and use Carmér's V [KT17] to compute the association between all the features and the target class for each dataset. Carmér's V is a typical measurement to compute the association of two categorical columns.

The results of this test are presented in Table 5.4 where we show the maximum Carmér's V value and the name of columns having this value for each dataset. As shown in the table, the Probe dataset contains two strongly dependent features ("flag" and "service") with the target class where both of them have a Carmér's V equal to 0.844. However, for other datasets, there are no fields with such strong dependencies to the target class. A strong association between two categorical columns means that the value of one column depends on the value of another column [JC]. Since we have strong association between the "service" or "flag" column and the target class, the value of the latter is related to the value of the former. LAFF is a form filling recommender system completely agnostic to anomaly detection; it learns the dependencies between the columns and use them to perform prediction of the target class. In the *Probe* dataset, LAFF uses the dependencies between "flag", "service", and the target class to perform prediction. Since these dependen-

Table 5.5: Results on Probe variations

	Prec	Rec
Probe	0.981	0.266
Probe-S	0.935	0.295
Probe-F	0.857	0.215
Probe-SF	1	1

cies are strong (due to the strong association), LAFF always returns a high confidence value for each prediction. This occurs even if the predicted value turns out to be anomalous. Hence, during the anomaly detection phase, the anomaly score is low, and the flag $checkProb_t$ in Algorithm 6 evaluates to false. As a result, LAFF-AD is not able to detect these anomalies.

In order to check if the strong dependency between "service" and "flag" features and the target class is the reason behind the inability of LAFF-AD to detect data anomalies (resulting in a low recall value), we ran LAFF-AD on four variations of the Probe dataset. We created the following datasets "Probe-S", "Probe-F", and "Probe-SF" representing the Probe dataset after removing the "service" column, the "flag column", and both of them from the dataset, respectively. The results of this experiment are presented in Table 5.5, where we computed the precision and recall of LAFF-AD on the four variations.

As shown in the table, removing only one of the columns "service" and "flag" does not affect the results of LAFF-AD. In fact, the results of LAFF-AD on Probe-s and Probe-F are quite similar to those obtained on the original Probe dataset. For the third variation, when we remove both columns, we can see that LAFF-AD can accurately detect all the anomalous instances (i.e., achieving a recall value of 1) with a precision value equal to 1. These results confirm our hypothesis that the strong association (dependency) with the "flag" and "service" columns leads LAFF to learn rare behaviors and predict the anomalous class with high confidence. This problem can be easily addressed during preprocessing, by removing columns that have degree of association higher than a certain threshold [MBGD14].

The answer to RQ1 is that LAFF-AD performs better than SOTA baselines on five datasets out of six used in our evaluation by at least 43.8 pp *and* 26 pp *in terms of precision and recall, respectively. LAFF-AD can detect at least 26% of data anomalies with a precision above 0.8.*

5.5.3 Performance (RQ2)

To answer RQ2, we measured the time needed to perform model training (considered as training time) and the time needed to decide anomalies (prediction time). Training time reflects the feasibility of using LAFF-AD in contexts where the training set is regularly updated with new instances. Prediction time indicates the ability of LAFF-AD to detect anomalies in a short period of time, for example as new data is acquired.

5.5.3.1 Methodology

In this RQ, we followed the same settings as RQ1, where we compared the time needed by LAFF-AD to train and predict with the same baselines that we used in RQ1. Training time measures the time to learn from the training data in order to detect data anomalies. Prediction time represents the average time needed by an algorithm to perform prediction for one suspicious instance. LAFF-AD's prediction time is measured as the sum of the prediction time taken by LAFF and the time taken by our anomaly detection algorithm.

5.5.3.2 Results

The results of this RQ are presented in the last two columns, *Train* and *Predict*, in Table 5.3. The *Train* column reports the training time in seconds whereas the *Predict* column contains two subcolumns indicating the average prediction time and the minimum/maximum time (in milliseconds).

As expected, training time for an algorithm varies from one dataset to another. LOF has the highest training time across datasets with a minimum of $9630\,\mathrm{s}$. EMAC-SCAN and iForest are the fastest algorithms: EMAC-SCAN does not require training while for iForest the training time is at most $1504\,\mathrm{s}$. As for LAFF-AD the training time is less than $7000\,\mathrm{s}$.

In terms of prediction time, LAFF-AD has the highest prediction time when compared to all the baselines. As we mentioned before, LAFF-AD's prediction time is the sum of the prediction time taken by LAFF and the time taken by our anomaly detection algorithm. As shown in Table 5.3, LAFF-AD takes on average at least $6\,\mathrm{ms}$ to perform prediction and at most $479\,\mathrm{ms}$. The reason behind the high prediction time for Covertype and CelebA is that the prediction relies on complex BNs. The complexity of the BN is defined in terms of the number of nodes (one node corresponds to one column) and the number of dependencies between different columns. As shown in Table 5.1, Covertype and CelebA have the highest number of columns among datasets.

These results need to be interpreted in our context. The training for the anomaly detection process is done offline and periodically. A training time of at most 7000 s is acceptable from a practical standpoint. This training time allows LAFF-AD to be trained daily if needed, especially when the training data is updated daily with thousands of instances.

Since anomaly detection is an offline process, a prediction time of at most 735 ms is fast enough. It can even enable online anomaly detection during the form filling process. Indeed, human-computer interaction standards [Hee00] indicate that a seamless interaction between a user and a data entry form can be ensured with a prediction time below 1 s. Since the prediction time proposed by LAFF-AD is comparable with the results of LAFF [BLBB22], this further confirms the possibility of using LAFF-AD to perform online data anomaly detection.

The answer to RQ2 is that the performance of LAFF-AD, with a training time below 7000 s *(less than 2 hours) and a prediction time of at most* 735 ms*, is suitable for practical applications. The training time of LAFF-AD usually lies between that of EMAC-SCAN and LOF. Concerning prediction time, LAFF-AD has a higher time compared to baselines but the difference has no practical implications since anomaly detection is an offline process.*

5.5.4 Threats to Validity

The size of the training sets is a common threat to all machine learning-based approaches. LAFF-AD is mainly designed to be used on datasets related to enterprise software systems, these datasets usually contain enough data for training LAFF-AD. In addition, these systems are often updated daily with thousands of new instances, which makes this limitation not particularly relevant.

To increase the generality of our results, LAFF-AD needs to be evaluated on different datasets from different domains. To deal with this issue, we evaluated LAFF-AD using benchmark datasets that have been previously used to evaluate anomaly detection approaches for categorical fields. Moreover, we selected datasets from different domains (e.g., biomedical, security, and finance). As shown in Table 5.1, these datasets also have different characteristics with respect to the number of rows, columns, and the range of categorical data. Also, we tried to evaluate LAFF-AD on different kinds of datasets, including datasets with synthetically-injected anomalies through our error injection strategy (i.e., the NCBI-E dataset) and benchmark datasets with real anomalies.

The implementation of the baselines can be considered an external threat. To minimize this threat, we used the official implementation of LOF and iForest in the `sklearn` library [PVG+11b]. As for EMAC-SCAN, we used the available implementation provided by Xu et al. [XWWW19]. All the scripts used to get the results were double-checked.

Another threat is the choice of parameter values of the baselines (i.e., iForest, OCSVM, and LOF). The value of the parameters depends mainly on the dataset because some values can work with one dataset but yield poor results on another dataset. In order to mitigate this threat, we checked the literature to identify possible ranges of values for each parameter in these baselines. We performed parameter tuning using grid search to select optimal parameter values for each baseline on each dataset.

5.6 Discussion

5.6.1 Usefulness

The main goal of LAFF-AD is to detect data anomalies in a suspicious data set. In order to evaluate the anomaly detection ability of LAFF-AD, we applied it on 6 datasets from different domains with varying characteristics, such as the number of rows and columns. Five out of six of these datasets are commonly used to evaluate existing anomaly detection methods and one of them (i.e., "NCBI-E") was used to evaluate LAFF, on which we build here.

Our results show that LAFF-AD outperforms different baselines on nearly all datasets (except Probe, see section 5.5.2) with a recall value of at least 0.570 and a precision higher than 0.808. In the context of anomaly detection, these results indicate that LAFF-AD can accurately detect at least 57% of the anomalous instances in the data, which is of practical significance. The high precision (Prec $\geq$ 0.808) also suggests a low number of *false alarms*). For some datasets (NCBI-E, Census, U2R, and CelebA), very high Recall imply that LAFF-AD can detect almost all the anomalies.

Regarding performance, LAFF-AD performs training in less than $6710\,\mathrm{s}$ (less than 2 hours) and has a prediction time of at most $735\,\mathrm{ms}$. These results suggests we can deploy LAFF-AD during the form-filling process as an additional step for data quality check. In other words, LAFF-AD can be useful to check if the data filled by the user is correct in real time. Further, applying LAFF-AD during the form-filling process can reduce the cost of fixing the detected anomalies [MBM15] because it can provide a list of suggested values directly to the user. Also, anomaly detection during form filling can prevent error propagation, since if errors may affect subsequent decisions if they are not quickly fixed [SE03].

5.6.2 Practical Implications

This subsection discusses the practical implications of LAFF-AD for different stakeholders: Data quality engineers, researchers, and software developers.

5.6.2.1 Data quality engineers

LAFF-AD is a data anomaly detection approach that can be easily used by data quality engineers. To be able to run LAFF-AD, data engineers need one training set and one dataset with suspicious instances that need to be checked. LAFF-AD does not require data engineers to tune any parameter.

5.6.2.2 Researchers

In this chapter, we repurpose a form-filling recommender system to detect data anomalies in categorical fields. To the best of our knowledge, our approach is the first approach that uses the output and characteristics of an automated form-filling tool like LAFF for anomaly detection. We speculate that our proposed solution can inspire researchers to use other recommender systems to extend our approach, or to repurpose similar automated form-filling tools for similar tasks such as anomaly detection for numerical fields.

5.6.2.3 Software Developers

Thanks to its short prediction time, LAFF-AD can be used as a data quality check step during the data entry process. After filling the fields in a form or a page, the user can run LAFF-AD to check the filled value before submission. Since LAFF-AD is based on a form-filling recommender system, LAFF-AD can be integrated as a stand-alone tool to perform online anomaly detection. Similar to LAFF, deploying LAFF-AD needs only a mapping between columns in the dataset and the data entry form fields. This mapping can be found in software documentation such as the database schema and the description of the UI widgets in the data entry forms [BLBB22].

5.7 Summary

In this chapter we proposed LAFF-AD, an approach to automatically detect data anomalies in categorical columns in offline datasets. LAFF-AD runs an adaptation of LAFF that handles offline prediction (i.e., not in real-time during the data entry process) to predict the value of a suspicious categorical field in the suspicious instance. LAFF-AD leverages the output of LAFF to detect data anomaly with a heuristic-based anomaly detection module.

We evaluated LAFF-AD using six datasets with different characteristics. Five of them are labeled datasets widely used to evaluate anomaly detection approaches for categorical data; the last dataset contains synthetic anomalies. The experimental results show that LAFF-AD can accurately detect a high range of data anomalies, with recall values between 0.6 and 1 and a precision value of at least 0.808. The results also show that LAFF-AD is fast enough to be applied to detect data anomalies in practice: LAFF-AD takes at most $7000\,\mathrm{s}$ and $735\,\mathrm{ms}$ to perform training and prediction respectively.

Chapter 6

Conclusions & Future Work

6.1 Conclusions

Data plays a fundamental role in modern systems such as software applications powered by machine learning. The effectiveness of such systems is affected by the quality of data used to make decisions. However, an important cause of low-quality data is data anomaly, which represents instances that deviate from the majority of data. These anomalies are mainly resulted from data entry errors (e.g., typographical errors and meaningless values). The goal of this thesis is to develop approaches to ensure data quality by preventing data entry errors during the form-filling process and by checking the offline data saved in databases.

In this thesis we have made the following contributions to achieve these goals:

1. *LAFF*: A learning-based automated approach for filling categorical fields in data entry forms. The approach utilizes Bayesian Networks to learn field dependencies from historical input instances. Moreover, LAFF relies on a clustering-based local modeling strategy to mine local field dependencies from partitions of historical input instances, to improve its learning ability. Furthermore, LAFF uses a heuristic-based endorser to ensure minimal accuracy for suggested values.

LAFF can provide a large number of accurate form filling suggestions, significantly outperforming state-of-the-art approaches in terms of Mean Reciprocal Rank (MRR). Further, LAFF takes at most 317 ms to provide a suggestion and is therefore applicable in practical data-entry scenarios.

2. *LACQUER*: An efficient learning-based automated approach for relaxing the completeness requirements of data entry forms. 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.

   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 $839\,\mathrm{ms}$ to provide a suggestion, which complies with human-computer interaction principles to ensure a seamless interaction with users.

3. *LAFF-AD*: An efficient anomaly detection tool to detect categorical data anomalies in offline datasets. LAFF-AD runs an adaptation of LAFF that handles offline prediction (i.e., not in real-time during the data entry process) to predict the value of a suspicious categorical field in the suspicious instance. LAFF-AD leverages the output of LAFF to detect data anomaly with a heuristic-based anomaly detection module.

   LAFF-AD can accurately detect a high range of data anomalies, with recall values between 0.6 and 1 and a precision value of at least 0.808. The results also show that LAFF-AD is fast enough to be applied to detect data anomalies in practice: LAFF-AD takes at most $7000\,\mathrm{s}$ and $735\,\mathrm{ms}$ to perform training and prediction respectively.

Contributions (1) and (2) focus mainly on preventing data entry errors during form-filling. Both approaches can be integrated into data entry forms as efficient and effective strategies to help the user during the form-filling process. Contribution (3) can be used offline on existing suspicious data to effectively detect categorical data anomalies.

6.2 Future Research Directions

This dissertation sets the basis to follow different research directions in the future:

Extension of LAFF. We plan to investigate methods to reduce the number of incorrect suggestions provided by LAFF when the number of filled fields used for prediction or the size of the training set is small.

Extension of LACQUER. We plan 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 [Roc, Mom, Goo] 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 Menu (Multi-select), Checkbox Group, etc.

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.

Extension of LAFF-AD. We plan to study the possibility of deploying LAFF-AD during the form filling process as an online anomaly detection technique for categorical fields.

User studies. We plan to conduct several empirical studies from the point of view of both users and developers to analyze the effect of *LAFF*, *LACQUER*, and *LAFF-AD* on reducing form filling time, input errors, and the cost of developing data entry forms.

Bibliography

[AA17] Charu C Aggarwal and Charu C Aggarwal. *An introduction to outlier analysis*. Springer, 2017.

[ABY16] Pierre A Akiki, Arosha K Bandara, and Yijun Yu. Engineering adaptive model-driven user interfaces. *IEEE Transaction on Software Engineering*, 42(12):1118–1147, 2016.

[ADK07] RB Aggarwal, Amit Dhawan, and Jay Shankar Kumar. Database-centric development of menus and graphic user interfaces. *Defence Science Journal*, 57(1):133, 2007.

[AFPS22] Fabrizio Angiulli, Fabio Fassetti, Luigi Palopoli, and Cristina Serrao. A density estimation approach for detecting and explaining exceptional values in categorical data. *Applied Intelligence*, pages 1–23, 2022.

[AGLH10] Samur Araujo, Qi Gao, Erwin Leonardi, and Geert-Jan Houben. Carbon: domain-independent automatic web form filling. In *Proc. ICWE'10*, volume 6189 of *LNCS*, pages 292–306, Berlin, Heidelberg, Germany, 2010. Springer, Springer Berlin Heidelberg.

[AHS12] Yuan An, Xiaohua Hu, and Il-Yeol Song. Learning to discover complex mappings from web forms to ontologies. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, pages 1253–1262, New York, NY, USA, 2012. ACM.

[AHWM20] Oluseun Omotola Aremu, David Hyland-Wood, and Peter Ross McAree. A machine learning approach to circumventing the curse of dimensionality in discontinuous time series machine data. *Reliability Engineering & System Safety*, 195:106706, 2020.

[AKV19] Stamatios-Aggelos N. Alexandropoulos, Sotiris B. Kotsiantis, and Michael N. Vrahatis. Data preprocessing in predictive data mining. *Knowledge Engineering Review*, 34:e1, 2019.

[AM09] Alnur Ali and Chris Meek. Predictive models of form filling. Technical Report MSR-TR-2009-1, Microsoft Research, January 2009. URL: https://www.microsoft.com/en-us/research/publication/predictive-models-of-form-filling/.

[Ame05] American Medical News. Data entry is a top cause of medication errors. https://amednews.com/article/20050124/profession/301249959/4/, 2005.

[And21] Android API Reference. Android view autofill. https://developer.android.com/reference/kotlin/android/view/autofill/package-summary, 2021.

[AP15] Ankur Ankan and Abinash Panda. Pgmpy: probabilistic graphical models using python. In *Proc. SCIPY'15*, pages 6–11, Austin, Texas, USA, 2015. SCIPY.

[APGG14] Iman Avazpour, Teerat Pitakrat, Lars Grunske, and John Grundy. Dimensions and metrics for evaluating recommendation systems. In *Recommendation Systems in Software Engineering*, pages 245–273. Springer, Berlin, Heidelberg, Germany, 2014.

[AR10] Atia M Albhbah and Mick J Ridley. Using ruleml and database metadata for automatic generation of web forms. In *2010 10th International Conference on Intelligent Systems Design and Applications*, pages 790–794. IEEE, 2010.

[AW12] Alexander Avidan and Charles Weissman. 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):173–181, 2012.

[AZ17] Maysoon Aldekhail and Djamal Ziani. Intelligent method for software requirement conflicts identification and removal: proposed framework and analysis. *International Journal of Computer Science and Network Security*, 17(12):91–95, 2017.

[Ban03] Bank for International Settlements. General guide to account opening and customer identification. https://www.bis.org/publ/bcbs85annex.htm, 2003.

[BBG11] Morten Bohøj, Niels Olof Bouvin, and Henrik Gammelmark. Adapforms: A framework for creating and validating adaptive forms. In *International Conference on Web Engineering*, pages 105–120. Springer, 2011.

[BCG+12] Tanya Barrett, Karen Clark, Robert Gevorgyan, Vyacheslav Gorelenkov, Eugene Gribov, Ilene Karsch-Mizrachi, Michael Kimelman, Kim D Pruitt, Sergei Resenchuk, Tatiana Tatusova, et al. Bioproject and biosample databases at NCBI: facilitating capture and organization of metadata. *Nucleic acids research*, 40(D1):D57–D63, 2012.

[BCL+21] Loli Burgueño, Robert Clarisó, Shuai Li, Sébastien Gérard, and Jordi Cabot. A nlp-based architecture for the autocompletion of partial domain models. In *Proc. CAiSE'21*, LNCS, Berlin, Heidelberg, Germany, 2021. Springer.

[BFSO84] Leo Breiman, Jerome Friedman, Charles J Stone, and Richard A Olshen. *Classification and regression trees*. CRC press, Boca Raton, Florida, USA, 1984.

[BKNS00] Markus M Breunig, Hans-Peter Kriegel, Raymond T Ng, and Jörg Sander. Lof: identifying density-based local outliers. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pages 93–104, 2000.

[BLBB22] Hichem Belgacem, Xiaochen Li, Domenico Bianculli, and Lionel Briand. A machine learning approach for automated filling of categorical fields in data entry forms. *ACM Trans. Softw. Eng. Methodol.*, apr 2022. Just Accepted. doi:10.1145/3533021.

[Bra97] Andrew P Bradley. The use of the area under the roc curve in the evaluation of machine learning algorithms. *Pattern recognition*, 30(7):1145–1159, 1997.

[BS03] Stephen D Bay and Mark Schwabacher. Mining distance-based outliers in near linear time with randomization and a simple pruning rule. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 29–38, 2003.

[CBHK02] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.

[CCC$^+$11] Kuang Chen, Harr Chen, Neil Conway, Joseph M Hellerstein, and Tapan S Parikh. Usher: Improving data quality with dynamic forms. *IEEE Transaction on Knowledge and Data Engineering*, 23(8):1138–1153, 2011.

[CCY00] Valeria Cardellini, Michele Colajanni, and Philip S Yu. Geographic load balancing for scalable distributed web systems. In *Proc. MASCOTS'00*, pages 20–27, San Francisco, CA, USA, 2000. IEEE.

[CFM02] Takeshi Chusho, Katsuya Fujiwara, and Keiji Minamitani. Automatic filling in a form by an agent for web applications. In *Proc. APSEC'02*, pages 239–247, Berlin, Heidelberg, Germany, 2002. IEEE.

[Coh95] William W Cohen. Fast effective rule induction. In *Machine learning proceedings 1995*, pages 115–123. Elsevier, 1995.

[CSFS02] Barbara Caputo, K Sim, Fredrik Furesjo, and Alex Smola. Appearance-based object recognition using svms: which kernel should i use? In *Proc of NIPS workshop on Statistical methods for computational experiments in visual processing and computer vision, Whistler*, volume 2002, 2002.

[CSY$^+$16] Tse-Hsun Chen, Weiyi Shang, Jinqiu Yang, Ahmed E Hassan, Michael W Godfrey, Mohamed Nasser, and Parminder Flora.

An empirical study on the practice of maintaining object-relational mapping code in Java systems. In *Proc. MSR'16*, pages 165–176, New York, NY, USA, 2016. ACM.

[CVM+21] Vendula Churová, Roman Vyškovskỳ, Kateřina Maršálová, David Kudláček, Daniel Schwarz, et al. Anomaly detection algorithm for real-world data and evidence in clinical research: implementation, evaluation, and validation study. *JMIR Medical Informatics*, 9(5):e27172, 2021.

[CVW11] Pablo Castells, Saúl Vargas, and Jun Wang. Novelty and diversity metrics for recommender systems: choice, discovery and relevance. In *Proc. DDR'11 - International Workshop on Diversity in Document Retrieval*, pages 29–36, http://www.dcs.gla.ac.uk/workshops/ddr2011/ddr2011.proceedings.pdf, 2011. self-published.

[DÁDS16] Isabel M Del Águila and José Del Sagrado. Bayesian networks for enhancement of requirements engineering: a literature review. *Requirements engineering*, 21:461–480, 2016.

[DOP13] Oscar Diaz, Itziar Otaduy, and Gorka Puente. User-driven automation of web form filling. In *Proc. ICWE'13*, volume 7977 of *LNCS*, pages 171–185, Berlin, Heidelberg, Germany, 2013. Springer.

[DS07] Kaustav Das and Jeff Schneider. Detecting anomalous records in categorical datasets. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 220–229, 2007.

[DSX10] Ofer Dekel, Ohad Shamir, and Lin Xiao. Learning to classify with missing and corrupted features. *Machine learning*, 81(2):149–178, 2010.

[DVB12] Karel Dejaeger, Thomas Verbraken, and Bart Baesens. Toward comprehensible software fault prediction models using bayesian network classifiers. *IEEE Transactions on Software Engineering*, 39(2):237–257, 2012.

[DVDSB+19] Fabiano Dalpiaz, Ivor Van Der Schalk, Sjaak Brinkkemper, Fatma Başak Aydemir, and Garm Lucassen. Detecting terminological ambiguity in user stories: Tool and experimentation. *Information and Software Technology*, 110:3–16, 2019.

[Ema99] Khaled El Emam. Benchmarking kappa: Interrater agreement in software process assessments. *Empirical Software Engineering*, 4(2):113–133, 1999.

[ER04] A Elbibas and MJ Ridley. Developing web entry forms based on metadata. In *International Workshop on Web Quality in conjunction with ICWE*. Citeseer, 2004.

[FGG97] Nir Friedman, Dan Geiger, and Moises Goldszmidt. Bayesian network classifiers. *Machine learning*, 29(2-3):131–163, 1997.

[FGG+12] Sergio Firmenich, Vincent Gaits, Silvia Gordillo, Gustavo Rossi, and Marco Winckler. Supporting users tasks with personal information management and web forms augmentation. In *Proc. ICWE2012*, volume 7387 of *LNCS*, pages 268–282, Berlin, Heidelberg, Germany, 2012. Springer, Oxford University Press.

[FGJ08] Wenfei Fan, Floris Geerts, and Xibei Jia. A revival of integrity constraints for data cleaning. *Proc. VLDB Endowment'08*, 1(2):1522–1523, 2008.

[Fie79] Stephen E Fienberg. The use of chi-squared statistics for categorical data problems. *Journal of the Royal Statistical Society: Series B (Methodological)*, 41(1):54–64, 1979.

[FS98] Martin R Frank and Pedro Szekely. Adaptive forms: an interaction technique for entering structured data. *Knowledge-Based Systems*, 11(1):37–45, 1998.

[FS04] Susan Fowler and Victor Stanwick. *Web application design handbook: Best practices for web-based software*. Morgan Kaufmann, Amsterdam, Boston, USA, 2004.

[Gaf20] Abdul Gafur. Updated tabular key and improved browser-based interactive key to species of pratylenchus filipjev (nematoda: Pratylenchidae). *Biodiversitas Journal of Biological Diversity*, 21(8):3780–3785, 2020.

[GB19] Jim Gee and Mark Button. The financial cost of fraud 2019: The latest data from around the world. 2019.

[GC06] Carl Gutwin and Andy Cockburn. Improving list revisitation with ListMaps. In *Proc. AVI'06*, pages 396–403, New York, NY, USA, 2006. ACM.

[GDBJ10] Mouzhi Ge, Carla Delgado-Battenfeld, and Dietmar Jannach. Beyond accuracy: evaluating recommender systems by coverage and serendipity. In *Proc. RecSys'10*, pages 257–260, New York, NY, USA, 2010. ACM.

[Ghe17] C. Ghezzi. Of software and change. *Journal of Software: Evolution and Process*, 29(9):e1888, 2017. e1888 smr.1888. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/smr.1888, arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/smr.1888, doi:https://doi.org/10.1002/smr.1888.

[GMH15] Baljinder Ghotra, Shane McIntosh, and Ahmed E Hassan. Revisiting the impact of classification techniques on the performance of defect prediction models. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, volume 1, pages 789–800. IEEE, 2015.

[GMP11] José A Gámez, Juan L Mateo, and José M Puerta. Learning bayesian networks by hill climbing: efficient methods based on progressive restriction of the neighborhood. *Data Mining and Knowledge Discovery*, 22(1-2):106–148, 2011.

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

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

[Goo08] Google. Chrome autofill forms. https://support.google.com/chrome, 2008. Accessed Feb 18, 2020.

[Haw80] Douglas M Hawkins. *Identification of outliers*, volume 11. Springer, 1980.

[HCH04] Bin He, Kevin Chen-Chuan Chang, and Jiawei Han. Discovering complex matchings across web query interfaces: a correlation mining approach. In *Proc. KDD'04*, pages 148–157, New York, NY, USA, 2004. ACM.

[HCT17] Zhe Hui Hoo, Jane Candlish, and Dawn Teare. What is an roc curve?, 2017.

[HDX05] Zengyou He, Shengchun Deng, and Xiaofei Xu. An optimization model for outlier detection in categorical data. In *Advances in Intelligent Computing: International Conference on Intelligent Computing, ICIC 2005, Hefei, China, August 23-26, 2005, Proceedings, Part I 1*, pages 400–409. Springer, 2005.

[HDXH06] Zengyou He, Shengchun Deng, Xiaofei Xu, and Joshua Zhexue Huang. A fast greedy algorithm for outlier mining. In *Advances in Knowledge Discovery and Data Mining: 10th Pacific-Asia Conference, PAKDD 2006, Singapore, April 9-12, 2006. Proceedings 10*, pages 567–576. Springer, 2006.

[Hee00] Carrie Heeter. Interactivity in the context of designed experiences. *J. of Interactive Advertising*, 1(1):3–14, 2000.

[HFH$^+$09] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. The weka data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1):10–18, 2009.

[HKOP03] Jan Horsky, David R Kaufman, Michael I Oppenheim, and Vimla L Patel. A framework for analyzing the cognitive complexity of computer-assisted clinical ordering. *Journal of Biomedical Informatics*, 36(1-2):4–22, 2003.

[HKTR04] Jonathan L Herlocker, Joseph A Konstan, Loren G Terveen, and John T Riedl. Evaluating collaborative filtering recommender systems. *ACM Transaction on Information Systems*, 22(1):5–53, 2004.

[HM09] Melanie Hartmann and Max Muhlhauser. Context-aware form filling for web applications. In *Proc. ICSC'09*, pages 221–228, Berkeley, CA, USA, 2009. IEEE.

[HRR19] Inma Hernández, Carlos R Rivero, and David Ruiz. Deep web crawling: a survey. *World Wide Web*, 22(4):1577–1610, 2019.

[HS94] L. A. Hermens and J. C. Shlimmer. A machine-learning apprentice for the completion of repetitive forms. *IEEE Expert*, 9(1):28–33, 1994.

[Hua98] Zhexue Huang. Extensions to the k-means algorithm for clustering large data sets with categorical values. *Data Mining and Knowledge Discovery*, 2(3):283–304, 1998.

[IPM16] Dino Ienco, Ruggero G Pensa, and Rosa Meo. A semisupervised approach to the detection and characterization of outliers in categorical data. *IEEE transactions on neural networks and learning systems*, 28(5):1017–1029, 2016.

[JBM11] Kawal Jeet, Nitin Bhatia, and Rajinder Singh Minhas. A bayesian network based approach for software defects prediction. *ACM SIGSOFT Software Engineering Notes*, 36(4):1–5, 2011.

[JC] LOX JELLY and Cream Cheese. Association between two categorical variables: Contingency analysis with chi square.

[JG09] Caroline Jarrett and Gerry Gaffney. *Forms that work: Designing Web forms for usability*. Morgan Kaufmann, Amsterdam, Boston, USA, 2009.

[JK19] Justin M Johnson and Taghi M Khoshgoftaar. Survey on deep learning with class imbalance. *Journal of Big Data*, 6(1):1–54, 2019.

[JK21] Faisal Jamil and Dohyeun Kim. An ensemble of prediction and learning mechanism for improving accuracy of anomaly detection in network intrusion environments. *Sustainability*, 13(18):10057, 2021.

[Jou19] Chichang Jou. Schema extraction for deep web query interfaces using heuristics rules. *Information Systems Frontiers*, 21(1):163–174, 2019.

[JQWX16] Xiao-Yuan Jing, Fumin Qi, Fei Wu, and Baowen Xu. Missing data imputation based on low-rank recovery and semi-supervised regression for software effort estimation. In *Proc. ICSE'16*, pages 607–618, Austin, TX, USA, 2016. IEEE.

[KB16] Marius Kaminskas and Derek Bridge. Diversity, serendipity, novelty, and coverage: a survey and empirical analysis of beyond-accuracy objectives in recommender systems. *ACM Transactions on Interactive Intelligent Systems*, 7(1):1–42, 2016.

[KCVM04] Trausti Kristjansson, Aron Culotta, Paul Viola, and Andrew McCallum. Interactive information extraction with constrained conditional random fields. In *Proc. AAAI'04*, volume 4, pages 412–418, New York, NY, USA, 2004. ACM.

[KJ10] Reza Khajouei and MWM Jaspers. The impact of cpoe medication systems' design aspects on usability, workflow and medication orders. *Methods of Information in Medicine*, 49(01):03–19, 2010.

[KJJ18] Mozhgan Karimi, Dietmar Jannach, and Michael Jugovac. News recommender systems–survey and roads ahead. *Information Processing & Management*, 54(6):1203–1227, 2018.

[KLZ$^+$19] Peter Kromkowski, Shaoran Li, Wenxi Zhao, Brendan Abraham, Austin Osborne, and Donald E Brown. Evaluating statistical models for network traffic anomaly detection. In *2019 systems and information engineering design symposium (SIEDS)*, pages 1–6. IEEE, 2019.

[KMH15] Gustavo Zanini Kantorski, Viviane Pereira Moreira, and Carlos Alberto Heuser. Automatic filling of hidden web forms: a survey. *ACM SIGMOD Record*, 44(1):24–35, 2015.

[KP17] Matevž Kunaver and Tomaž Požrl. Diversity in recommender systems–a survey. *Knowledge-based Systems*, 123:154–162, 2017.

[KSZ08] Hans-Peter Kriegel, Matthias Schubert, and Arthur Zimek. Angle-based outlier detection in high-dimensional data. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 444–452, 2008.

[KT17] Jalayer Khalilzadeh and Asli DA Tasci. Large sample size, significance level, and the effect size: Solutions to perils of using big data for academic research. *Tourism Management*, 62:89–96, 2017.

[LIJ+19] Hoyeop Lee, Jinbae Im, Seongwon Jang, Hyunsouk Cho, and Sehee Chung. Melu: Meta-learned user preference estimator for cold-start recommendation. In *Proc. SIGKDD'19*, pages 1073–1082, New York, NY, USA, 2019. ACM.

[LTZ12] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation-based anomaly detection. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 6(1):1–39, 2012.

[LVMPG14] Mario Linares-Vásquez, Collin McMillan, Denys Poshyvanyk, and Mark Grechanik. On using machine learning to automatically classify software applications into domain categories. *Empirical Software Engineering*, 19(3):582–618, 2014.

[LZPQ18] Junli Li, Jifu Zhang, Ning Pang, and Xiao Qin. Weighted outlier detection of high-dimensional categorical data using feature grouping. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 50(11):4295–4308, 2018.

[MBGD14] Caitlin Mills, Nigel Bosch, Art Graesser, and Sidney D'Mello. To quit or not to quit: predicting future behavioral disengagement from reading patterns. In *Intelligent Tutoring Systems: 12th International Conference, ITS 2014, Honolulu, HI, USA, June 5-9, 2014. Proceedings 12*, pages 19–28. Springer, 2014.

[MBM+11] Tim Menzies, Andrew Butcher, Andrian Marcus, Thomas Zimmermann, and David Cok. Local vs. global models for effort estimation and defect prediction. In *Proc. ASE'11*, pages 343–351, Lawrence, KS, USA, 2011. IEEE.

[MBM15] Kıvanç Muşlu, Yuriy Brun, and Alexandra Meliou. Preventing data errors with continuous testing. In *Proc. ISSTA'15*, pages 373–384, New York, NY, USA, 2015. ACM.

[Mic13] Microsoft. Change the default tab order for controls on a form. `https://support.microsoft.com/en-us/office/change-the-default-tab-order-for-con` trols-on-a-form-03d1599a-debf-4b66-a95b-e3e744210afe, 2013.

[MK17a] Ruchika Malhotra and Megha Khanna. An empirical study for software change prediction using imbalanced data. *Empirical Software Engineering*, 22(6):2806–2851, 2017.

[MK17b] S. McIntosh and Y. Kamei. Are fix-inducing changes a moving target? a longitudinal case study of just-in-time defect prediction. *IEEE Transaction on Software Engineering*, 44(5):412–428, 2017.

[MK19] Elinore F McCance-Katz. The national survey on drug use and health: 2017. *Substance abuse and mental health services administration*, page 7, 2019.

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

[MPFN18] Emilia Mendes, Mirko Perkusich, Vitor Freitas, and João Nunes. Using bayesian network to estimate the value of decisions within the context of value-based software engineering. In *Proceedings of the 22nd international conference on evaluation and assessment in software engineering 2018*, pages 90–100, 2018.

[MROE$^+$19] Marcos Martínez-Romero, Martin J O'Connor, Attila L Egyedi, Debra Willrett, Josef Hardi, John Graybeal, and Mark A Musen. Using association rule mining and ontologies to generate metadata recommendations from multiple biomedical databases. *Database J. Biol. Databases Curation*, 2019, 2019.

[NBS12] Keith Noto, Carla Brodley, and Donna Slonim. Frac: a feature-modeling approach for semi-supervised and unsupervised anomaly detection. *Data mining and knowledge discovery*, 25:109–133, 2012.

[NK08] Kazuyo Narita and Hiroyuki Kitagawa. Detecting outliers in categorical record databases based on attribute associations. *Lecture Notes in Computer Science*, 4976:111–123, 2008.

[OEDK18] John-Paul Ore, Sebastian Elbaum, Carrick Detweiler, and Lambros Karkazis. Assessing the type annotation burden. In *Proc. ASE'18*, pages 190–201, New York, NY, USA, 2018. ACM.

[OY14] Ahmet Okutan and Olcay Taner Yıldız. Software defect prediction using bayesian networks. *Empirical Software Engineering*, 19:154–181, 2014.

[PCC16] Guansong Pang, Longbing Cao, and Ling Chen. Outlier detection in complex categorical data by modeling the feature value couplings. 2016.

[PCC21] Guansong Pang, Longbing Cao, and Ling Chen. Homophily outlier detection in non-iid categorical data. *Data Mining and Knowledge Discovery*, pages 1–62, 2021.

[PCJ$^+$17] Spencer Pearson, José Campos, René Just, Gordon Fraser, Rui Abreu, Michael D Ernst, Deric Pang, and Benjamin Keller. Evaluating and improving fault localization. In *Proc. ICSE'17*, pages 609–620, Buenos Aires, Argentina, 2017. IEEE.

[PLM15] Sebastian Proksch, Johannes Lerch, and Mira Mezini. Intelligent code completion with bayesian networks. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 25(1):1–31, 2015.

[POKB20] Tahereh Pourhabibi, Kok-Leong Ong, Booi H Kam, and Yee Ling Boo. Fraud detection: A systematic literature review of graph-based anomaly detection approaches. *Decision Support Systems*, 133:113303, 2020.

[PTAJ16] Guansong Pang, Kai Ming Ting, David Albrecht, and Huidong Jin. Zero++: Harnessing the power of zero appearances to detect anomalies in large-scale data sets. *Journal of Artificial Intelligence Research*, 57:593–620, 2016.

[PVG$^+$11a] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[PVG$^+$11b] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al.

Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830, 2011.

[QMR$^+$20] Siyu Qian, Esther Munyisia, David Reid, David Hailey, Jade Pados, and Ping Yu. Trend in data errors after the implementation of an electronic medical record system: A longitudinal study in an australian regional drug and alcohol service. *International Journal of Medical Informatics*, 144:104292, 2020.

[RA20] Seyed Ehsan Roshan and Shahrokh Asadi. Improvement of bagging performance for classification of imbalanced datasets using evolutionary multi-objective optimization. *Engineering Applications of Artificial Intelligence*, 87:103319, 2020.

[Raf95] Adrian E Raftery. Bayesian model selection in social research. *Sociological Methodology*, 25:111–163, 1995.

[RHH11] Lida Rashidi, Sattar Hashemi, and Ali Hamzeh. Anomaly detection in categorical datasets using bayesian networks. In *Artificial Intelligence and Computational Intelligence: Third International Conference, AICI 2011, Taiyuan, China, September 24-25, 2011, Proceedings, Part II 3*, pages 610–619. Springer Berlin Heidelberg, 2011.

[RJMFSC23] Santiago del Rey Juárez, Silverio Juan Martínez Fernández, and Antonio Salmerón Cerdán. Bayesian network analysis of software logs for data-driven software maintenance. *IET Software*, pages 1–19, 2023.

[RNDL$^+$08] Enrico Rukzio, Chie Noda, Alexander De Luca, John Hamard, and Fatih Coskun. Automatic form filling on mobile devices. *Pervasive and Mobile Computing*, 4(2):161–181, 2008.

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

[RSMMA$^+$19] NNR Ranga Suri, Narasimha Murty M, G Athithan, NNR Ranga Suri, Narasimha Murty M, and G Athithan. Outlier detection in categorical data. *Outlier Detection: Techniques and Applications: A Data Mining Perspective*, pages 69–93, 2019.

[Rus10] Stuart J Russell. *Artificial intelligence a modern approach*. Pearson Education, Inc., 2010.

[RWZ09] Martin Robillard, Robert Walker, and Thomas Zimmermann. Recommendation systems for software engineering. *IEEE Software*, 27(4):80–86, 2009.

[SAF22] Halima Sadia, Syed Qamar Abbas, and Mohammad Faisal. A bayesian network-based software requirement complexity prediction model. In *Computational Methods and Data Engineering: Proceedings of ICCMDE 2021*, pages 197–213. Springer, 2022.

[SAM18] Amr Rekaby Salama, Ozge Alaçam, and Wolfgang Menzel. Text completion using a context-integrating dependency parser. In *Proc. RepL4NLP'18*, pages 41–49, Melbourne, Australia, 2018. ACL.

[SE03] David Saff and Michael D Ernst. Reducing wasted development time via continuous testing. In *14th International Symposium on Software Reliability Engineering, 2003. ISSRE 2003.*, pages 281–292. IEEE, 2003.

[SGS18] Qinbao Song, Yuchen Guo, and Martin Shepperd. A comprehensive investigation of the role of imbalanced learning for software defect prediction. *IEEE Transactions on Software Engineering*, 45(12):1253–1269, 2018.

[SHBA$^+$14] Mirjam Seckler, Silvia Heinz, Javier A Bargas-Avila, Klaus Opwis, and Alexandre N Tuch. Designing usable web forms: empirical evaluation of web form improvement guidelines. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1275–1284, 2014.

[SLDM18] Rasmus Strømsted, Hugo A López, Søren Debois, and Morten Marquard. Dynamic evaluation forms using declarative modeling. *BPM (Dissertation/Demos/Industry)*, 2196:172–179, 2018.

[SMA12] NNR Ranga Suri, M Narasimha Murty, and Gopalasamy Athithan. An algorithm for mining outliers in categorical data through ranking. In *2012 12th International Conference on Hybrid Intelligent Systems (HIS)*, pages 247–252. IEEE, 2012.

[SMA13] NNR Ranga Suri, Musti Narasimha Murty, and Gopalasamy Athithan. A rough clustering algorithm for mining outliers in categorical data. In *Pattern Recognition and Machine Intelligence: 5th International Conference, PReMI 2013, Kolkata, India, December 10-14, 2013. Proceedings 5*, pages 170–175. Springer, 2013.

[SR12] Marton Sakal and Lazar Rakovic. Errors in building and using electronic tables: Financial consequences and minimisation techniques. *Strategic Management*, 17(3):29–35, 2012.

[STL11] Gunnar Schröder, Maik Thiele, and Wolfgang Lehner. Setting goals and choosing metrics for recommender system evaluations. In *UCERSTI2 workshop at the 5th ACM conference on recommender systems*, volume 23, page 53, New York, NY, USA, 2011. ACM.

[SZ03] Andrew Sears and Ying Zha. Data entry for mobile devices using soft keyboards: Understanding the effects of keyboard size and user tasks. *J. of Human-Computer Interaction*, 16(2):163–184, 2003.

[SZZZ17] Jiuling Song, Yonghe Zhou, Juren Zhang, and Kewei Zhang. Structural, expression and evolutionary analysis of the non-specific phospholipase c gene family in gossypium hirsutum. *BMC genomics*, 18(1):1–15, 2017.

[TB96] Paul Thistlewaite and Steve Ball. Active forms. *Computer Networks and ISDN Systems*, 28(7-11):1355–1364, 1996.

[TBA17] Luigi Troiano, Cosimo Birtolo, and Roberto Armenise. Modeling and predicting the user next input by bayesian reasoning. *Soft Computing*, 21(6):1583–1600, 2017.

[TCdSdM10] Guilherme A Toda, Eli Cortez, Altigran S da Silva, and Edleno de Moura. A probabilistic approach for automatically filling form-based web interfaces. *Proc. of the VLDB Endowment*, 4(3):151–160, 2010.

[TH19] Ayman Taha and Ali S Hadi. Anomaly detection methods for categorical data: A review. *ACM Computing Surveys (CSUR)*, 52(2):1–35, 2019.

[TLA$^+$22] Florian Tambon, Gabriel Laberge, Le An, Amin Nikanjam, Paulina Stevia Nouwou Mindom, Yann Pequignot, Foutse

Khomh, Giulio Antoniol, Ettore Merlo, and François Laviolette. How to certify machine learning based safety-critical systems? a systematic literature review. *Automated Software Engineering*, 29(2):38, 2022.

[ULI19] Qasim Umer, Hui Liu, and Inam Illahi. Cnn-based automatic prioritization of bug reports. *IEEE Transaction on Reliability*, 69(4):1341–1354, 2019.

[VDBB08] Antal Van Den Bosch and Toine Bogers. Efficient context-sensitive word completion for mobile devices. In *Proc. MobileHCI'08*, pages 465–470, New York, NY, USA, 2008. ACM.

[VLL+03] Costas Vassilakis, Giorgos Laskaridis, Giorgos Lepouras, Stathis Rouvas, and Panagiotis Georgiadis. A framework for managing the lifecycle of transactional e-government services. *Telematics and Informatics*, 20(4):315–329, 2003.

[W3C17] W3CSchools. HTML dom input text object. `https://www.w3schools.com/jsref/dom_obj_text.asp`, 2017.

[W3C21] W3C School. Html ¡input¿ autocomplete attribute. `https://www.w3schools.com/tags/att_input_autocomplete.asp`, 2021.

[WBL+13] Johanna I Westbrook, Melissa T Baysari, Ling Li, Rosemary Burke, Katrina L Richardson, and Richard O Day. The safety of electronic prescribing: manifestations, mechanisms, and rates of system-related errors associated with two commercial systems in hospitals. *Journal of the American Medical Informatics Association*, 20(6):1159–1167, 2013.

[WC21] Zeqing Wu and Weishen Chu. Sampling strategy analysis of machine learning models for energy consumption prediction. In *2021 IEEE 9th International Conference on Smart Energy Grid Engineering (SEGE)*, pages 77–81. IEEE, 2021.

[WGV+11] Marco Winckler, Vicent Gaits, Dong-Bach Vo, Firmenich Sergio, and Gustavo Rossi. An approach and tool support for assisting users to fill-in web forms with personal information. In *Proc. SIGDOC'11*, pages 195–202, New York, NY, USA, 2011. ACM.

[WKL04] Wan MN Wan-Kadir and Pericles Loucopoulos. Relating evolving business rules to software design. *Journal of Systems Architecture*, 50(7):367–382, 2004.

[WZK+14] Shaohua Wang, Ying Zou, Iman Keivanloo, Bipin Upadhyaya, Joanna Ng, and Tinny Ng. Automatic reuse of user inputs to services among end-users in service composition. *IEEE Transaction on Services Computing*, 8(3):343–355, 2014.

[WZNN17] Shaohua Wang, Ying Zou, Joanna Ng, and Tinny Ng. Context-aware service input ranking by learning from historical information. *IEEE Transaction on Services Computing*, 14(1):97–110, 2017.

[XWC+18] Hongzuo Xu, Yongjun Wang, Li Cheng, Yijie Wang, and Xingkong Ma. Exploring a high-quality outlying feature value set for noise-resilient outlier detection in categorical data. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, pages 17–26, 2018.

[XWWW19] Hongzuo Xu, Yongjun Wang, Zhiyue Wu, and Yijie Wang. Embedding-based complex feature value coupling learning for detecting outliers in non-iid categorical data. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 5541–5548, 2019.

[YBL14] Xin Ye, Razvan Bunescu, and Chang Liu. Learning to rank relevant files for bug reports using domain knowledge. In *Proc. FSE'14*, pages 689–699, New York, NY, USA, 2014. ACM.

[YLX+16] Xinli Yang, David Lo, Xin Xia, Lingfeng Bao, and Jianling Sun. Combining word embedding with information retrieval to recommend similar bug reports. In *Proc. ISSRE'16*, pages 127–137, Ottawa, ON, Canada, 2016. IEEE.

[YS20] Li Yang and Abdallah Shami. On hyperparameter optimization of machine learning algorithms: Theory and practice. *Neurocomputing*, 415:295–316, 2020.

[YSY+20] Junwen Yang, Utsav Sethi, Cong Yan, Alvin Cheung, and Shan Lu. Managing data constraints in database-backed web applications. In *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*, pages 1098–1109. IEEE, 2020.

[Zho21] Zhi-Hua Zhou. Ensemble learning. In *Machine Learning*, pages 181–210. Springer, Singapore, 2021.

[ZZW19] Mingrui Ray Zhang, Shumin Zhai, and Jacob O. Wobbrock. Text entry throughput: towards unifying speed and accuracy in a single performance metric. In *Proc. CHI'19*, pages 1–13, New York, NY, USA, 2019. ACM.