Université Jean Monnet

Le Laboratoire d'Analyse et de Modélisation de Systèmes pour l'Aide à la Décision (LAMSADE), Université Paris Dauphine

Machine Learning and Data Mining (MLDM)

Master's 1 Internship Report

Assisting Data Curator Using Pre-Trained ML Models

Author:

Ismail BACHCHAR

Under the supervisions of:

Dr. Khalid Belhajjame

Prof. Grigori Daniela

Academic Tutor:

Prof. Amaury Habrard

Paris, France - April, 2022 to August, 2022

Dedication

"To my lovely family..."

Acknowledgments

I would like to express my very great appreciation to my advisors Dr. Khalid Belhajjame, Prof. Grigori Daniela, and Prof. Amaury Habrard for their valuable and constructive suggestions during the planning and development of this research work. their willingness to give their time so generously has been very much appreciated. I would also thank the LAMSADE laboratory at Paris Dauphine University (`www.lamsade.dauphine.fr`) for providing me with the opportunity to do this research internship.

Abstract

There is a wide range of domain-specific techniques to assess and improve data quality in the literature. Unfortunately, these solutions primarily target data that resides in relational databases and data warehouses. Even though the data quality is crucial in the Machine Learning (ML) field as it dramatically impacts the models' efficiency, accuracy, and complexity. There are still few efforts towards improving the data quality over enhancing the predictive power of ML models. In this work, we propose a method of assisting the data quality curation task using an interpretable pre-trained model proven effective, namely Decision Tree. In doing so, we try to identify the key features that represent data quality issues, therefore, impact the predictive power of the effective model. And we propose a system that pinpoints corrupted data values within the data instances that are the cause behind false predictions made by the model.

Keywords: Data Quality; Data Quality Assessment, Decision Tree, Machine Learning

Contents

List of Tables

List of Figures

Chapter 1

Introduction and Context of the Project

The presented work in this report was held at Le Laboratoire d'Analyse et de Modélisation de Systèmes pour l'Aide à la Décision (LAMSADE), Université Paris Dauphine (`www.lamsade.dauphine.fr`) under the supervision of Dr. Khalid Belhajjame, and Prof. Grigori Daniela with whome I was supported and oriented during my internship's period at the laboratory.

The initial subject of this internship was Capturing and Using Provenance in Machine Learning Pipelines, that is intending to assist Machine Learning (ML) practitioners in debugging ML models by capturing their artifacts during different phases, such as in data pre-processing, training, and evaluation phases. In doing so, it would be easy for the ML developer to debug: detecting and removing existing and potential code and model errors. For this latter, an example scenario would be when the developer gets unexpected results (e.g. false predictions) from the model and wants to fix it without touching the input data. In this case, s/he might analyze the model's generated artifacts in every phase to understand why the model gives such results. Maybe analyze the artifacts generated from the pre-processing phase in order to verify that the data is being manipulated/transformed as expected after every operation. In such a scenario, the artifacts would be the newly created data after every operation of the pre-processing steps.

In such a saturated research field, plenty of powerful tools are on the market to solve exactly the same problem. Such as the system developed by Amazon [1], which is able to extract, store and manage metadata and provenance information in ML experiments in the three most used ML libraries SparkML, Scikit-Learn, and MXNet. Another powerful system is the TensorFlow Debugger (tfdbg) [2], which is developed by Google, the developer of TensorFlow itself. This system is a whole package to debug ML pipelines. Even if it's specialized for TensorFlow, it remains a powerful tool in the sense that it provides all the debugging functionalities, such as breakpoints on the calculation graph and easy-to-understand visualizations of the pipeline operations and data lineage. Also, there are the MLOps toolkits such as the Data Version Control (DVC), Pachyderm, and much

more that can capture, save and manipulate the pipeline provenance and data. Lastly are Notebooks-based tools, an open-source web application that creates interactive documents, including all types of writings such as codes, text, and media (video, images, ..). It supports a wide range of programming languages, and it is used by millions of developers. For such a widely used tool, there are plenty of tools to solve the provenance capturing problem. Firstly, in [3], the authors provided a mechanism to capture, save and analyze the provenance of Python scripts inside the IPythonNotebooks by the integration with noWorkflow [4]. Secondly, a more recent approach in [5] tries to convert notebooks into workflows where notebook developers need to follow a set of guidelines in writing code. These two approaches require the developer to integrate some code into their own and are only based on the Python programming language, limiting their use. To solve these two limits, another approach is to create a separate plugin for the Notebook, so there is no need to change the developer's scripts or learn how to use a new tool. As ProvBook [6], that is an extension of Jupyter Notebook to capture and track the provenance of the Notebook cells over time. For every cell, the tool is able to track and save the cell's run history with its code and metadata, such as the execution time. Another attempt is in [7], where the authors introduce the ML-ProvLab as a JupyterLab -an open-source development environment for Jupyter Notebooks- extension that can automatically identify the relationships between data and models in ML scripts. Then it tracks, captures, compares, and visualizes the provenance of machine learning notebooks. All these tools are very powerful in capturing the ML pipeline's provenance regardless of how they do it, if it is time-consuming or requires enormous memory, and regardless of the tools and languages, they support.

For this reason, the internship's objective was shifted toward a fresh research field that still can provide value to the research community and focuses on helping to accelerate the work of Data Scientists and ML practitioners. Data Scientists spend an important amount of their time dealing with data challenges before coming to the model creation. As for Data Collection, the developer needs to define and manage different data sources to acquire the coming data and store it in an easily accessible manner. Another important challenge is Data Preprocessing, which is an important technique to convert raw data into clean data sets that are reliable and exploitable.
Even though ML practitioners spend most of their time in this last step, they still miss detecting corrupted data values -non-logical values (e.g, outliers)- which is crucial for the model's performance and requires a linear examination of every value in the dataset. The presented work aims to assist Data Scientists in curating their datasets by pointing them to the Data Quality issues based on pre-trained models.

Breif research on Data Quality concerns:

Data quality is a concern in numerous domains. For example, in Machine Learning, the effectiveness of prediction models critically depends on the quality of the data used in building the models [8]. Data quality assessment also plays a critical role in evaluating the usefulness of data collection, especially in Software engineering domains, such as in the Team Software Process frameworks [9] and empirical software engineering research [10].

Data quality is studied in numerous other domains including, but not limited to, cyber-

physical systems [11], assisted living systems [12], citizen science [13], ERP systems [14], accounting information systems [15], drug databases [16], smart-cities [17], sensor data streams [18], linked data [19], data integration [20], [21], multimedia data [22], scientific workflows [23], and customer databases [24]. Bigdata management [25], Internet of Things (IoT) [26], and machine learning [27] domains are generating renewed interest in data quality research. A wide range of domains specific techniques to assess and improve the quality of data exist in the literature [28], [29], [30]. Lack of data quality in these domains manifests in several forms, including missing, incomplete, inconsistent, inaccurate, duplicate, and dated data.

Chapter 2

Methodological Contribution

It is well understood from the literature that any ML model's efficiency, accuracy, and complexity depend on the quality of the available data. This is more enforced by the fact that Data Scientists and ML practitioners spend more and more of their time on data preparation tasks, including loading and cleaning data. The pre-processing stage is required for every ML problem as the data remains susceptible to errors or irregularities that may get introduced during this stage or even earlier. These kinds of errors are hard to detect by humans as they sometimes require scanning the whole dataset points. For example, the non-logical values - not relevant to the problem - are very hard to detect as they need the Data scientist to go through the whole data values looking for the ones that do not make sense to the problem. In addition, there is the problem of leaked errors that might occur even after the pre-processing steps, such as a human error in the annotation stage.

There are plenty of ways to handle data quality issues for ML problems, such as using software monitoring tools and dashboards. In this work, we concentrate on a specific scenario where data quality issues might be solved using a pre-trained model known to be accurate and efficient. We consider a case where the Data Scientist wants to use a machine learning model M that is, as said, proven to be accurate. However, the robust model might be provided by other parties that do not want to share its construction and only offer its prediction power. For example, a medical scientist may wish to use a pre-trained model that fellow scientists produced to make predictions about the patients they supervise. In such a context, the user will likely not have access to the dataset used to train the model nor the pipeline used to pre-process the data. This scenario is even more prevalent in contexts where the dataset used for training comprise sensitive information about individuals, such as health condition and income. The Data Scientist has access only to the ML model's properties, such as the features metadata (accepted formats and types), set of classes if it is a classification problem, etc. However, the scientist might obtain extra information about the model if possible. For example, A Decision-Tree model gives information about the number of used data points in the training set.

Given a dataset D and an accurate model M, the objective is to use the prediction power of M to detect data quality issues in D. The Data Scientists might do some pre-processing on D if needed. For example, aligning the features with what is accepted by

the model M, e.g. input shape and type etc. However, the data quality issues are not guaranteed to disappear; features might still suffer from numerous problems that will not be detected nor repaired. Applying the model M directly to the dataset D is more likely to generate false predictions for some (if not most) of the observations in D. Therefore, data quality errors would need to be detected and repaired before placing confidence in the results generated by the model. But to do so, the Data Scientist must use their knowledge of the dataset D when making predictions to know when the model makes bad predictions. Because these wrongly predicted observations are the ones used to pinpoint feature values that suffer from the data quality issues. Furthermore, notice that not all the features in D have the same impact on the model's prediction performance. Some of them have little or even no effect. In this case, it is preferable to focus the curation task, which involves human intervention, on the essential features of the problem at hand.

With the above in mind, we examine, in this work, the problem of assisting the curation task by identifying the key features that impact the quality of the prediction made by the model M and pinpointing the feature values of a given observation that may suffer from data quality issues and are the cause behind an erroneous prediction. To do so, we leverage a source of information that is readily available, namely the model M. This problem can be broken down into the following sub-problems.

2.1 Identification of the features to be curated

Given a machine learning model M that was trained on a non-accessible dataset DS_t, and given a new dataset DS with the same features as required by M, in other words $features(DS) = features(DS_t)$, identify the subset of features $FS \subseteq features(DS)$ that needs to be curated for data quality issues to improve the quality of the predictions obtained when applying M to the observations in DS. The following function can define this problem:

Problem 1

$$keyFeatures : ML \rightarrow \mathcal{P}(features(DS))$$

Here, we seek to focus the curator's attention on a subset of features in the dataset rather than trying to curate all features since a number of these features may not impact the predictions made by M. Our approach is consistent with the *quality with a purpose* maxim.

Different variants of the problem just stated can be defined. For example, the user might only be interested in making predictions on a particular set of classes for a classification problem. Hence, they do not need the full predictive power of the model M. In this case, we can identify which features should be retained to get curated after and which can be ignored by the curator. Those must-keep features are a subset of the complete set of features used by the model M. Therefore, by focusing on such a subset, the curator will spend less time repairing data quality issues irrelevant to the task at hand.
The following function can define this problem, where C designates the set of interest classes.

Problem 1.1

$$keyFeatures : ML \times \mathcal{P}(C) \rightarrow \mathcal{P}(features(DS))$$

Another variant of the *Problem* 1 would focus on identifying the features that need to be curated to obtain predictions with certain accuracy. The underlying assumption of this variant is that the quality of the predictions made by the model is proportionate to the cost of addressing data quality issues in DS. For example, suppose the user is satisfied with making predictions with a certain accuracy threshold (e.g. 90%). This can reduce the number of must-curate features to achieve the accuracy target. This problem can be defined by the following function, where A is an interval from 1 to 100 representing the user's required accuracy threshold.

Problem 1.2

$$keyFeatures : ML \times A \rightarrow \mathcal{P}(features(DS))$$

We now turn to the second problem, in which we seek to help the user identify the feature values that might be causing incorrect predictions.

2.2 Error Detection

Given an observation, $o \in DS'$, for which the prediction obtained using the model M is different from the ground truth prediction, i.e., $pred(o) \neq gt(o)$, identify the feature values in o that are likely to be responsible for the erroneous prediction, if any.
Note that in our wording of the problem, we state "*if any*". This is because the prediction error may be due to the model itself and not the data. That said, we assume that the used model has proven effective. Furthermore, some models produce, in addition to the predicted class, the confidence of the prediction, which can be harnessed to focus on examining the observations that are likely to suffer from data quality issues. As well as pinpointing the feature values that are likely to suffer from data quality errors, we may use the introspection of the model to assist the curator by stating conditions that need to be satisfied as a result of the data repair. Such conditions will guide the user in repairing the observation's values.

The way in which we can address the previously stated problems depends on the nature of the used model. In particular, interpretable models are more likely to be relatively straightforward to work with, while non-interpretable models are likely to be less helpful. Therefore, in the Experiments and Results 3 chapter, we choose to work with a widely used interpretable model, namely Decision Tree.

Chapter 3

Experiments and Results

In this section, we will explain the proposed method in Methodological Contribution 2 for handling data quality issues using Decision Tree as the pre-trained model. Using a pre-trained model, in general, helps put the causes of any false prediction on the data itself. Even though that is not the case sometimes, we suppose to be in our case. This way, to detect the data quality issues within the input data, we are at least sure that every not-well predicted observation has some data issue within its values. The problem is half solved when we are sure it is not because of the model. In that manner, we can entirely focus on the input data and connect its false prediction with its input. For this latter, we suggest using an interpretable model that is easy to understand and directly draws the relationship between the input and output data (predictions) to answer the question of what to change in the input to change the output. In our case, we chose to work with the Decision Tree model as we could not think of other more interpretable models that link observation values and predictions.

In what follows, we explain the chosen Decision Tree (DT) model, how we solve the previously discussed problems in the case of using DT as the pre-trained model, then the experiments setup and a brief discussion of the obtained results.

3.1 Decision Tree as The Pre-Trained Model

The Decision Tree is one of the most known and developed machine learning methods. It is often used in data mining, and business intelligence for prediction and diagnostic tasks [31, 32]. It is also used in classification problems, regression problems, and time-dependent prediction. The main strength of DT induction is its interpretability characteristics. It is a graphical method for problems involving a sequence of decisions and successive events. More precisely, its results formalize the reasoning that an expert could have to reproduce the series of decisions and find the characteristic of an object. The main advantage of this model type is that a human being can easily understand and produce decision sequences to predict the target category of a new instance. The results provide a graphic structure or a base of rules that facilitates understanding and corresponds to human reasoning.

Learning by Decision Tree is part of supervised learning, where the class of each object in the database is given. The goal is to build a model from a set of examples associated with the classes to find a description for each class from the common properties between the instances. Once this model has been built, we can extract a set of classification rules. In this model, the extracted rules are then used to classify new objects whose class is unknown. The prediction procedure travels a path from the root to a leaf. The class returned (default class) is the most frequent among the examples on the sheet. At each tree's internal node (decision node), there is a test (question) corresponding to an attribute in the learning base and a branch corresponding to each of the attribute's possible values. At each leaf node, there is a class value/label. Therefore, a path from the root node to another node corresponds to a series of attributes (questions) and values (answers). This flowchart-like structure with recursive partitioning helps the user in decision-making. It is this visualization that easily mimics human-level thinking. That is why Decision Trees are easy to understand and interpret.

Another advantage of Decision Tree induction is its ability to automatically identify the most discriminating features for a use case, i.e., the most representative data inputs for a given task. Its flexibility and autonomy explain this as a model with little assumption on the hypothesis space. It is an approach that remains particularly useful for input space problems and a powerful tool able to handle large-scale problems, thus particularly useful in big data mining. However, it is generally less accurate than other machine learning models like neural networks.

In brief, this learning algorithm has the following three essential characteristics:

1. **Interpretability:** Because of its flowchart-like structure, the way attributes interact to give a prediction is very readable.
2. **Efficiency:** The induction process is done by a top-down algorithm which recursively splits terminal nodes of the current tree until they all contain elements of only one class. Practically, the algorithm is very fast in terms of running time and can be used on extensive datasets (e.g. millions of objects and thousands of features).
3. **Flexibility:** This method does not make any hypothesis about the problem under consideration. It can handle both continuous and discrete attributes. Predictions at leaf nodes may be symbolic or numerical (in which case, trees are called regression trees).

Now we give a formal definition of the Decision Tree model that corresponds to our previously discussed problems:

A Decision Tree $DT = (N, E)$ is a binary tree, where an internal node is characterized by the tuple.

$$n = (\#obs, att, p, L, c_n)$$

where $\#obs$ is the number of observations that are associated with the node n.
p is a boolean predicate of the form $att \leq \lambda$, with λ a numerical threshold.

L is a list specifying the number of observations in $\#obs$ that are associated with each class $c_i \in C$.
c_n is the prevalent class within the observations associated with n.

The leaf nodes of a DT are characterized by a subset of the attributes describing the internal nodes. More precisely, a leaf node is not associated with an attribute or a predicate.

Every internal node n in DT has two edges $e_p, e_{\neg p} \in E$ associating it with its children. e_p associates n with a child node representing the observations satisfying the predicate p, and $e_{\neg p}$ associates n with a child node representing the observations not satisfying the predicate p.

3.2 Pruning of Decision Tree Models

Pruning Decision Tree follows a model selection procedure where the models are the pruned subtrees of the original tree -the bigger tree. This procedure minimizes a penalized criterion where the penalty is proportional to the number of leaves in the tree [33]. Defining a Decision Tree's optimal size consists of stopping the pre-pruning or reducing the post-pruning of the tree to have the best-pruned sub-tree from the original tree in the sense of a generalization error. Broadly, this generalization error consists of improving the predictive aspect of the tree, on the one hand, and reducing its complexity, on the other hand. To this end, several pruning methods have been developed, such as:

- **Minimal cost complexity pruning (MCCP):** is also called post-pruning for the CART algorithm. This method consists of constructing a nested sub-tree sequence using a formulation called minimum cost complexity, which is incredibly detailed in [31].
- **Reduced error pruning (REP):** estimates the real error of a given subtree on a pruning or test set. The pruning algorithm is performed as follows: "As long as there is a tree that a leaf can get replaced without increasing the estimate of the real error, then prune this tree". This technique gives a slightly congruent tree in the sense that some examples may be misclassified. The study of Elomaa and Kääriäinen [34] presents a detailed analysis of the REP method. In this analysis, the two authors evoke that the REP method was introduced by Quinlan [35], but the latter never presents it algorithmically, which is a source of confusion. Even though REP is considered a straightforward, almost trivial algorithm for pruning, many different algorithms have the same name. There is no consensus on whether REP is a bottom-up algorithm or an iterative method. Moreover, it is not apparent that the training or pruning set is used to determine the labels of the leaves that result from pruning.
- **Pessimistic error pruning (PEP):** To overcome the disadvantages of the REP method, [35] proposed a pruning strategy that uses a single construction set and pruning of the tree. The tree is pruned by examining the error rate at each node and

assuming that the true error rate is considerably worse. For example, if a given node contains N records in which E among them are misclassified, then the error rate is estimated at E=N. The central concern of the algorithm is to minimize this estimate by considering this error rate as a very optimistic version of the real error rate [36].

- **Minimum error pruning (MEP):** was proposed by Niblett and Bratko [37], critical value pruning (CVP) by Mingers [38] and error-based pruning (EBP) proposed by Quinlan as an improvement of the PEP method for the C4.5 algorithm.

3.3 Problem Solutions in the Case of Decision Tree

In the following subsections, we show the proposed solutions for every previously discussed problem in section 2 in the case of using a Decision Tree as the pre-trained model.

3.3.1 Key Feature Identification Using Decision Tree

In the case of using a Decision Tree as the pre-trained model, the problem of identifying the key features (a subset of features of the original set of variables) introduced in 2.1 can be resolved in a straightforward manner. In particular, the features to curate are the features that are used in the internal nodes of the Decision Tree. Depending on the problem, those key features might be the whole set of features or just the important ones that are repeatedly used throughout the DT model. Specifically, they can be defined by the following comprehension set.

$$KeyFeatures(DT) \ = \ \{n.att \ s.t. \ n \in DT.N\}$$

The other variants of the same problem, namely problems 1.1 *and* 1.2, are related to the pruning of the Decision Tree model [39] which was briefly discussed in 3.2. However, while the state-of-the-art techniques in this field aim to reduce the tree size to avoid overfitting and/or to make the tree readable and easily understandable by a human, our goal is instead to reduce the number of features that the Decision Tree model uses. As such, such techniques are not directly applicable to our problem and, as such, have to be adapted to cater for the specifics of our problems. In what follows, we provide for each of these variants a solution that would be used.

Problem 1.1:
In the case when using a DT-based model, given a set of classes of interest, $C' \subset C$ where C is the set of all features. The user only cares about making predictions/classifications on those classes. And the goal is to find the key features that support this constraint so the user can reduce their time while doing data quality curation. This problem has a straightforward solution in the case of using the DT model. Because of the fact that every prediction is a path that goes from the DT root node to one of its leaf nodes. Finding the key features is strictly keeping the features that appear in every prediction's path with a leaf node labeled with a class $c \in C'$. As a result of this solution, the Decision Tree needs to be pruned so that we end up with a sub-tree of the original one while only keeping the

paths that link the root with the leaf nodes labeled with a class in C'.

Problem 1.2:
In this problem, the user seeks to focus the curation task on a subset of the features while keeping the model's accuracy at a specific value. We can apply existing pruning methods [40] that reduce accuracy error when pruning the Decision Tree. For example, a method that is used for this purpose would be to prune the Decision Tree in an iterative manner, where in every iteration, the node n that is pruned is the one that minimizes the following error:

$$err = \frac{err(pruneTree(DT, t)) - err(DT)}{|leaves(DT)| - |leaves(pruneTree(DT, t))|}$$

Our objective here is different, however, in the sense that we mainly aim to reduce the number of features. For example, the Decision Tree obtained using the above method may result in a small tree size (sub-tree) but uses all of the original Decision Tree's features in a mixed manner. We could add another constraint in each algorithm iteration to avoid this problem. This constraint is to choose the node n that minimizes the above error and reduces the number of features used by one feature, that is:

$$|features(pruneTree(DT, t)| - |features(DT)|$$

The algorithm iterates until it reaches a point when pruning any node n yields an accuracy below the threshold λ specified by the user.

3.3.2 Error Detection Using Decision Tree

In this part, we focus on detecting the corrupted data values within a given observation using a Decision Tree pre-trained model for classification problems. The objective is to detect the observation's values representing potential data quality issues. For example, if there is a normalized feature and some of its values are not normalized. The goal here is to recommend those values to the user to check if they are considered data quality issues.

This solution's procedure starts with extracting the Decision Tree information from all of its nodes in a structured way. Then for every observation that is falsely classified, we suggest several possible data quality issues within the observation's values that must be curated in order for the observation to be well classified.

For the coding part of this proposed solution, we choose to use the python Scickit-Learn library as it provides a tree-based structure of the Decision Tree model. Furthermore, it is a very known library within the Machine Learning community as millions of ML developers use it.

Given a pre-trained DC model and a falsely classified data point -an observation that might have a data quality issue within its values- alongside its ground truth class label

-known by the user-, we solve the error detection problem in the following two steps:

1) Model's data extraction:

In Scikit-Learn, the DT classifier is represented as a binary tree structure as several parallel arrays accessible through the attribute *tree_*, which also holds some other low-level attributes such as *node_count*, the total number of nodes, and the maximal tree depth. Those parallel arrays hold information about every node in the tree. For instance, the i-th element of each array holds data about node i -node 0 is the tree's root. Some of the arrays only apply to either leaf or split nodes. In this case, the values of the nodes of the other type are arbitrary. For example, the arrays feature and threshold only apply to split nodes. The values for leaf nodes in these arrays are, therefore, arbitrary. Among these arrays, there are:

- $children\_left[i]$: id of the left child of node i or -1 if leaf node
- $children\_right[i]$: id of the right child of node i or -1 if leaf node
- $feature[i]$: feature used for splitting node i
- $threshold[i]$: threshold value at node i
- $n\_node\_samples[i]$: the number of training samples reaching node i
- $impurity[i]$: the impurity at node i

In our case, we gather three types of data. First, the $leaf\ nodes'\ data$, which is obtained by propagating every internal node's, including the root, thresholds to its children. While doing so, we only keep track of the minimum and maximum feature thresholds. For example, at the root node, the threshold of feature A is 100. Subsequently, at one of this node's children, the same feature has a small threshold, such as 50. In this case, we only keep the small threshold as we know that for feature A to be valid -to satisfy- at this node, it must have a value less than 50. Therefore, the procedure is to keep doing this propagation of thresholds until reaching the leaf nodes. To end up with several features reaching the leaf nodes with some inequalities that any observation values must verify in order for it to be in one of those leaves as shown in the 3.1. Second, the $leaf\ nodes'\ metadata$, which is precisely all nodes' information. For every node, we store all its information, namely impurity, samples, value, and class label. Last, the $feature\ thresholds$, when we store the thresholds of every feature of the problem at hand throughout the tree.
After gathering all this data and storing it in JSON format files, we apply the next step, which uses it as the basis of searching for erroneous values within the observation's values.

2) Get all possible data quality issues of an observation:

As we know, every leaf node has several feature inequalities, as depicted in Figure 3.1 and is associated with a class label. Therefore, we want to group those class labels with their leaf nodes while keeping, for every leaf node, the features that are not satisfied by the observation values. As a result, we end up with a dictionary D such as: {'0': {'6': ['2',

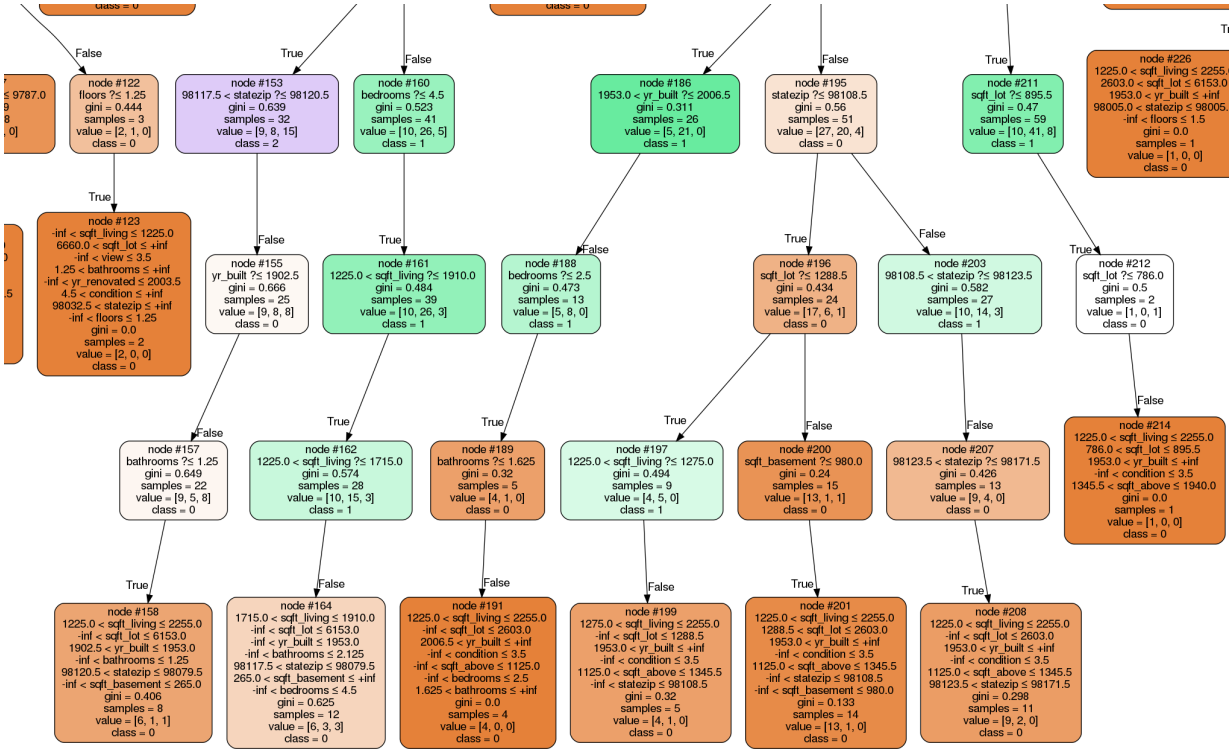

Figure 3.1: *Example of a Decision (sub)Tree with leaf nodes inequalities*

'3'], '9': ['2', '3', '8'] } , ...} : leaf nodes 6 and 9 belonging to class 0 (labeled with class 0) have different features that are not satisfied by the observation's values. Satisfaction holds when the observation's value at the same feature satisfies the feature's inequality ([min max] interval).

Using the prediction path -the sequence of nodes starting from root to leaf nodes-, we get the set of features PF that satisfy all the inequalities of the fall-in leaf node. Moreover, given the ground truth class label, we can access all its associated leaf nodes using the previously obtained dictionary D. Every node in D holds a set of feature inequalities not satisfied by the observation's values, o. However, suppose we make value changes to any observation feature (value) to satisfy the feature inequality of any node. In that case, it is like changing the prediction path to make the observation fall in that leaf node. Therefore, keeping only the features at the intersection of PF and those in every node in D. We get the possible changes -equivalent to the erroneous values in o- that could correct the classification by changing one or multiple values in o.

To illustrate this solution, we use a house price dataset [41]. Even though it is often used in regression problems, we twist it to fit our purpose, which is to classify house prices into three categories: class '0' represents low-priced houses, '1' medium, and '2' the expensive ones. We use this dataset in particular because it wrangles a large set of property sales records with unknown data quality issues. Moreover, the dataset consists of the real estates markets such as those in Sydney and Melbourne. The dataset consists of 4600 observations and 14 features that are described as follows:

- **'bedrooms':** the number of available bedrooms in the house.
- **'bathrooms':** the number of available bathrooms in the house.
- **'sqft_living':** the living area of the house.
- **'sqft_lot':** the maximum house size.
- **'floors':** the levels of the house.
- **'waterfront':** A binary value representing if the house is next to an area of water.
- **'view':** The number of house views.
- **'condition':** The proximity to the main road or railroad.
- **'sqft_above':** Refers to all living square feet in the house above the ground.
- **'sqft_basement':** The height of the basement.
- **'yr_built':** The built year of the house.
- **'yr_renovated':** The renovated year of the house.
- **'output':** Represents the house price category (0, 1 and 2).
- **'statezip':** The state zip code

Before applying the proposed solution, some pre-processing was applied on the data to handle the most known issues, such as:

- Lexical errors, e.g., typos and spelling mistakes.
- Irregularities, e.g., abnormal data values and data formats.
- Violations of the Integrity constraint.
- Outliers.
- Duplications.
- Missing values.
- Inconsistency, e.g., inhomogeneity in values and types in representing the same data.

After that, the data became clean for the training, and we use the same cleansed data in order to detect the other not handled data quality issues. A data view of this dataset is presented in Figure 3.2.

This part aims to find the corrupted data values within a given observation. We propose a system that takes the pre-trained model M -Decision Tree in this case-, a falsely classified observation o, and the ground truth class label. We extract the model's data as described

	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	sqft_above	sqft_basement	yr_built	yr_renovated	statezip	output
0	3.0	1.50	1340	7912	1.5	0	0	3	1340	0	1955	2005	98133	0
1	5.0	2.50	3650	9050	2.0	0	4	5	3370	280	1921	0	98119	2
2	3.0	2.00	1930	11947	1.0	0	0	4	1930	0	1966	0	98042	0
3	3.0	2.25	2000	8030	1.0	0	0	4	1000	1000	1963	0	98008	1
4	4.0	2.50	1940	10500	1.0	0	0	4	1140	800	1976	1992	98052	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
4595	3.0	1.75	1510	6360	1.0	0	0	4	1510	0	1954	1979	98133	0
4596	3.0	2.50	1460	7573	2.0	0	0	3	1460	0	1983	2009	98007	1
4597	3.0	2.50	3010	7014	2.0	0	0	3	3010	0	2009	0	98059	1
4598	4.0	2.00	2090	6630	1.0	0	0	3	1070	1020	1974	0	98178	0
4599	3.0	2.50	1490	8102	2.0	0	0	4	1490	0	1990	0	98042	0

4600 rows × 14 columns

Figure 3.2: *The used dataset view*

		Feature name	Value	Satisfied	Must Satisfy
Node	**Feature**				
65	**3**	sqft_lot	11374.000000	[6660.0, None]	[None, 6660.0]
377	**2**	sqft_living	128.432785	[None, 1225.0]	[2155.0, 2255.0]
644	**2**	sqft_living	128.432785	[None, 1225.0]	[3610.0, None]
610	**2**	sqft_living	128.432785	[None, 1225.0]	[3725.0, 3756.5]
655	**2**	sqft_living	128.432785	[None, 1225.0]	[3145.0, None]
...	**...**	...	...	...	...
422	**11**	yr_renovated	0.000000	[None, 2003.5]	[1998.5, None]
242	**2**	sqft_living	128.432785	[None, 1225.0]	[1225.0, 2255.0]
	3	sqft_lot	11374.000000	[6660.0, None]	[2603.0, 6153.0]
	12	statezip	98168.000000	[98032.5, None]	[98032.5, 98036.0]
	7	condition	5.000000	[4.5, None]	[None, 4.5]

Figure 3.3: *The list of possible changes*

first in 3.3.2 1 and then present a list of possible changes as previously discussed in 3.3.2 2, meaning the changes that the user can adapt in order for the observation to be correctly re-classified. Those changes as depicted in Figure 3.3 include features that could be edited/curated by the user to change the prediction path toward another leaf node labeled by the actual class. This way, the possible changes might contain one feature change (curating one feature value of the observation) or multiple features. As shown in Figure 3.3, every row represents a change of the observation's original value (column $Value$) from the satisfied leaf node inequality (column $Satisfied$), which gives the false classification, to satisfy another leaf node inequality that is labeled with the actual class (column $MustSatisfy$). The number of required feature changes orders the changes list. There are changes of one feature presented at the top and multiple features below it, such as the 4 features change in the last row, in the same Figure 3.3.

At this point, the objective is to look for suspicious changes in the list of all possible changes. We define a suspicious change as a change that has an illogical $Value$, which means a value that does not make sense with regard to the leaf node's inequality bounds (the $Satisfied$ column) or among all the values of the Decision Tree model (all the model's tree thresholds) of the same feature. Those suspicious changes will be ranked first based on a calculated score β. This score is a positive value calculated for every change represented in the mentioned list in order to order it and suggest first the changes of small score values (β). The β score is calculated by taking the division of the observation's original value ($Value$ column) and its closest value of the set of values extracted from either the satisfied leaf node ($Satisfied$ column) or the whole Decision Tree thresholds -all the model's decisions- of the identical change's feature ($feature$ column). This division represents how close the observation's value is to either a leaf node inequality bounds or all the DT model's thresholds of the same change's feature. The more the score's value is close to zero, the more confident there is a data quality issue within that feature's value of the observation.

In the following subsections, we present some obtained results using the M model and randomly choosing an observation from the dataset that is falsely classified alongside its ground truth label, then change one of its features' values to an arbitrary value. The goal is to see if we can detect this change if the new value is considered illogical in regards to the feature's all values.

1) Using only the satisfied leaf node inequalities:
For this approach, we use only the satisfied leaf node inequality bounds to calculate the β score and then sort the changes by the most negligible score value for every change as presented in Figure 3.4. As it shows, the changes are ordered by the small value of β, which indicates that the observation's value ($Value$ column) is not coherent with the leaf node inequality. Thus it might represent a data quality issue. For example, the first row shows that the observation value 128.43 of the feature $sqft\_living$ is not coherent with the $Satisfied$ leaf node inequality bounds, $[-\infty, 1225]$ ($None$ is equivalent to $-\infty$). Therefore, we consider this value a data quality issue, and it is presented at the top of the list for the user to curate it. Another helpful piece of information is the *suggested times* column which is the number of times that the exact change, with the same values, is repeated or suggested. The user could also look at this information to get an in-depth understanding

Node	Feature	Feature name	Value	Satisfied	Must Satisfy	score	suggested times
610	2	sqft_living	128.432785	[None, 1225.0]	[3725.0, 3756.5]	0.034479	1
644	2	sqft_living	128.432785	[None, 1225.0]	[3610.0, None]	0.035577	1
655	2	sqft_living	128.432785	[None, 1225.0]	[3145.0, None]	0.040837	2
576	2	sqft_living	128.432785	[None, 1225.0]	[2255.0, 3145.0]	0.056955	2
377	2	sqft_living	128.432785	[None, 1225.0]	[2155.0, 2255.0]	0.059598	1
371	2	sqft_living	128.432785	[None, 1225.0]	[2012.5, 2155.0]	0.063818	2
332	2	sqft_living	128.432785	[None, 1225.0]	[1225.0, 2012.5]	0.104843	2
65	3	sqft_lot	11374.000000	[6660.0, None]	[None, 6660.0]	0.585546	1
91	1	bathrooms	2.000000	[1.25, None]	[None, 1.25]	0.625000	2
123	4	floors	1.500000	[1.25, None]	[None, 1.25]	0.833333	1
119	7	condition	5.000000	[4.5, None]	[None, 4.5]	0.900000	3

Figure 3.4: *One-feature changes list ordered by the score (using only leaf node inequality)*

of the suggestions.

After that, we calculate the list of suggested changes of multiple-features, and order them by the $Total\ score$ which is calculated by the sum of every feature's score within the same change. For instance, in Figure 3.5, the first row (first suggested change) says that changing the $sqft\_living$ value from 128.43 to another value in the $Must\ Satisfy$ interval [2255, 3145], and also changing the $yr\_renovated$ value from 0 to a value in [2008, 2012.5] will make the observation goes to the leaf node number 438 which is labeled as the actual class. Thus correcting the classification. However, our goal is not to correct the classification but instead to detect the erroneous values within the observation which are the $sqft\_living$ and $yr\_renovated$ values in this example. We follow the same procedure to sort the other changes of multiple-features as they are presented in the appendix, Figures A.1 and A.2.

2) Using all the Decision Tree model thresholds:

Using the same previously described procedure, we sort the list of suggested changes of one-feature by the calculated score, *score*, and then the multiple-feature changes by the $Total\ score$. However, instead of using only the $Satisfied$ leaf node inequality, we use all the Decision Tree thresholds extracted in the section 3.3.2. In Figure 3.6 we see the same previously suggested feature $sqft\_living$ suggested again here with a high number of repetitions, *suggested times*. This indicates that the $sqft\_living$ feature suffers from data quality issues and needs to get curated. This latter is enforced by the fact that the value of this feature is the one that was changed in the experiment. Another example is shown in Figure 3.7 where the same previously discussed features, $sqft\_living$ and $yr\_renovated$, are again proposed at the top.

At the end of this, we generated 331 experiments using the two approaches, using only satisfied leaf node inequality and all the DT thresholds. In each one, we generate

Node	Feature	Feature name	Value	Satisfied	Must Satisfy	score	suggested times	Total score
438	2	sqft_living	128.432785	[None, 1225.0]	[2255.0, 3145.0]	0.056955	1	0.056955
	11	yr_renovated	0.000000	[None, 2003.5]	[2008.0, 2012.5]	0.000000	1	0.056955
591	2	sqft_living	128.432785	[None, 1225.0]	[3145.0, 3725.0]	0.040837	2	0.525686
	1	bathrooms	2.000000	[1.25, None]	[4.125, 4.375]	0.484848	2	0.525686
414	2	sqft_living	128.432785	[None, 1225.0]	[2255.0, 3145.0]	0.056955	5	0.710067
	3	sqft_lot	11374.000000	[6660.0, None]	[None, 7428.5]	0.653112	5	0.710067
126	1	bathrooms	2.000000	[1.25, None]	[1.25, 1.625]	0.812500	1	0.812500
	11	yr_renovated	0.000000	[None, 2003.5]	[2003.5, None]	0.000000	1	0.812500
328	2	sqft_living	128.432785	[None, 1225.0]	[1225.0, 2012.5]	0.104843	25	1.103682
	12	statezip	98168.000000	[98032.5, None]	[98049.5, 98054.0]	0.998839	25	1.103682
95	3	sqft_lot	11374.000000	[6660.0, None]	[6660.0, 7222.5]	0.635001	3	1.260001
	1	bathrooms	2.000000	[1.25, None]	[None, 1.25]	0.625000	3	1.260001
34	3	sqft_lot	11374.000000	[6660.0, None]	[None, 6660.0]	0.585546	12	1.585032
	12	statezip	98168.000000	[98032.5, None]	[98097.0, 98117.5]	0.999486	12	1.585032
110	1	bathrooms	2.000000	[1.25, None]	[None, 1.25]	0.625000	1	1.623360
	12	statezip	98168.000000	[98032.5, None]	[None, 98007.0]	0.998360	1	1.623360
116	3	sqft_lot	11374.000000	[6660.0, None]	[6660.0, 9741.0]	0.856427	1	1.756427
	7	condition	5.000000	[4.5, None]	[None, 4.5]	0.900000	1	1.756427

Figure 3.5: *Two-feature changes list ordered by the Totalscore (using only leaf node inequality)*

Node	Feature	Feature name	Value	Satisfied	Must Satisfy	score	suggested times
377	2	sqft_living	128.432785	[None, 1225.0]	[2155.0, 2255.0]	0.144307	11
119	7	condition	5.000000	[4.5, None]	[None, 4.5]	0.900000	3
65	3	sqft_lot	11374.000000	[6660.0, None]	[None, 6660.0]	0.965360	1
91	1	bathrooms	2.000000	[1.25, None]	[None, 1.25]	1.000000	2
123	4	floors	1.500000	[1.25, None]	[None, 1.25]	1.000000	1

Figure 3.6: *One-feature changes list ordered by the score (using all DT thresholds)*

Node	Feature	Feature name	Value	Satisfied	Must Satisfy	score	suggested times	Total score
438	2	sqft_living	128.432785	[None, 1225.0]	[2255.0, 3145.0]	0.144307	1	0.144307
	11	yr_renovated	0.000000	[None, 2003.5]	[2008.0, 2012.5]	0.000000	1	0.144307
126	1	bathrooms	2.000000	[1.25, None]	[1.25, 1.625]	1.000000	1	1.000000
	11	yr_renovated	0.000000	[None, 2003.5]	[2003.5, None]	0.000000	1	1.000000
414	2	sqft_living	128.432785	[None, 1225.0]	[2255.0, 3145.0]	0.144307	1	1.109666
	3	sqft_lot	11374.000000	[6660.0, None]	[None, 7428.5]	0.965360	1	1.109666
328	2	sqft_living	128.432785	[None, 1225.0]	[1225.0, 2012.5]	0.144307	1	1.144296
	12	statezip	98168.000000	[98032.5, None]	[98049.5, 98054.0]	0.999990	1	1.144296
591	2	sqft_living	128.432785	[None, 1225.0]	[3145.0, 3725.0]	0.144307	1	1.144307
	1	bathrooms	2.000000	[1.25, None]	[4.125, 4.375]	1.000000	1	1.144307
116	3	sqft_lot	11374.000000	[6660.0, None]	[6660.0, 9741.0]	0.965360	1	1.865360
	7	condition	5.000000	[4.5, None]	[None, 4.5]	0.900000	1	1.865360
34	3	sqft_lot	11374.000000	[6660.0, None]	[None, 6660.0]	0.965360	1	1.965349
	12	statezip	98168.000000	[98032.5, None]	[98097.0, 98117.5]	0.999990	1	1.965349
95	3	sqft_lot	11374.000000	[6660.0, None]	[6660.0, 7222.5]	0.965360	1	1.965360
	1	bathrooms	2.000000	[1.25, None]	[None, 1.25]	1.000000	1	1.965360
110	1	bathrooms	2.000000	[1.25, None]	[None, 1.25]	1.000000	1	1.999990
	12	statezip	98168.000000	[98032.5, None]	[None, 98007.0]	0.999990	1	1.999990

Figure 3.7: *Two-feature changes list ordered by the Totalscore (using all DT thresholds)*

the one and multiple-feature changes and some metadata about the experiment, such as the original randomly chosen observation, its ground truth class, and the predicted one. This last experiment aims to determine the features that are always presented in the top three suggested changes in every experiment of the 331 ones. As a result, Table 3.1 illustrates, for every feature, the number of times it is presented in the top 3 suggestions with different numbers of features within the suggested change when using only the satisfied leaf node inequality. For example, the $waterfront$ feature was not included in any of the suggested changes. This is because this feature is not essential for the classification problem. Therefore, it was not used throughout the tree -there are few to no nodes using the $waterfront$ feature to split on-. Another example is the $sqft\_lot$ feature, which is presented in the top 3 suggestions of one-feature changes in 64 experiments of the 331 ones -notice that each experiment picks a random observation from the original data and changes, randomly, one of its features-. In table 3.2 the same result is shown when using all the Decision Tree thresholds. We can see that the most used features are identical to those found by the first approach (when using only the satisfied leaf node inequality values).

Table 3.1: Number of presented features in the top-3 change suggestions when using only the satisfied leaf node inequality values

	1-feature	2-features	3-features	4-features	5-features	6-features
waterfront	0	0	0	0	0	0
sqft_lot	64	61	65	44	5	0
yr_built	51	49	54	54	15	0
view	18	19	22	3	0	0
condition	10	7	9	5	0	0
bedrooms	3	4	4	1	0	0
sqft_basement	0	0	0	0	0	0
bathrooms	19	20	18	4	0	0
sqft_above	15	15	18	13	0	1
yr_renovated	2	2	2	1	0	2
sqft_living	14	15	15	8	1	0
statezip	6	8	7	0	1	0
floors	0	0	0	0	0	0

Table 3.2: Number of presented features in the top-3 change suggestions when using all the DT thresholds

	1-feature	2-features	3-features	4-features	5-features	6-features
waterfront	0	0	0	0	0	0
sqft_lot	64	66	66	44	5	0
yr_built	51	47	53	27	16	1
view	19	22	22	3	0	0
condition	10	11	9	8	0	0
bedrooms	3	4	4	1	0	0
sqft_basement	0	0	0	0	0	0
bathrooms	19	19	19	4	1	0
sqft_above	15	15	16	13	0	1
yr_renovated	2	5	3	1	0	2
sqft_living	15	15	15	8	1	0
statezip	6	8	8	2	1	0
floors	0	0	0	0	0	0

Chapter 4

Conclusion and Future Work

The presented work addresses a vital research field: data quality issues for machine learning tasks. In our approach, we proposed a different way of assisting in curating those issues. We imagine a scenario where the user has access to a powerful ML model to make predictions but has no idea about the model-building phases. Therefore, the user does not even know the pre-processing steps to do in order to make predictions with their dataset. In such a case, the user could take advantage of the model's effectiveness by considering every false predicted observation is related to the input data and not the model. Doing so lets the user fucose on curating their dataset's possible quality issues while understanding what pre-processing techniques they might apply.

In this work, we used the Decision Tree (DT) as the pre-trained model, and we proposed two problems alongside their solutions in the case of using DT. First, identifying the features to be curated, which aims at suggesting to the user the important features they need to pay more attention to as they guide the predictions. For this problem, we proposed two other variants that introduce some constraints, namely the accuracy constraint when the user does not want to use the total capacity of the model, and maybe they agree with some accuracy threshold to detect quality issues; and the classes of interest constraint, when the user wants to identify data quality issues within some class labels of the problem (in the case of classification). Second, error detection, in the case of using Decision Tree, we proposed a system that takes the pre-trained DT, a falsely classified observation, and its ground truth class label as input and provides the user with a list of all possibles changes to make in order to correct the classification which is precisely equivalent to the detection of data quality issues because the observation's values in every change are the ones indicating a quality issue.

Improvements and Future Work:
In this work, we still lack a lot of improvements to create a fully controlled system that assists the users in doing data curation tasks faster. Up to now, we have been mainly using python scripts for the coding part [42] and some Jupyter Notebooks to manipulate suggested data quality issues. Still, as for our target audience for this work, we must propose a final product with a web-based application, maybe, that helps accelerate the curation process. Also, this work is not well generalized as it still misses some functionalities

such as the generalization of using more models, not only the Decision Tree, and addressing more problems, not only classification ones. Lastly, in this work, we couldn't have the possibility to work on real-world data as it is the only way to evaluate the outcome of this work. So we, instead, worked on a publically available dataset and a traditional problem. In future work, we will continue developing this work, especially the generalization to use more models and creating a final product that could be used by anyone interested in data curation.

Bibliography

[1] S. Schelter, J.-H. Böse, J. Kirschnick, T. Klein, and S. Seufert, "Automatically tracking metadata and provenance of machine learning experiments," in *NeurIPS 2017*, 2017. [Online]. Available: https://www.amazon.science/publications/automatically-tracking-metadata-and-provenance-of-machine-learning-experiments

[2] S. Cai, E. Breck, E. Nielsen, M. Salib, and D. Sculley, "Tensorflow debugger: Debugging dataflow graphs for machine learning," 2016.

[3] J. F. N. Pimentel, V. Braganholo, L. Murta, and J. Freire, "Collecting and analyzing provenance on interactive notebooks: When IPython meets noWorkflow," in *7th USENIX Workshop on the Theory and Practice of Provenance (TaPP 15)*. Edinburgh, Scotland: USENIX Association, Jul. 2015. [Online]. Available: https://www.usenix.org/conference/tapp15/workshop-program/presentation/pimentel

[4] J. a. F. Pimentel, L. Murta, V. Braganholo, and J. Freire, "Noworkflow: A tool for collecting, analyzing, and managing provenance from python scripts," *Proc. VLDB Endow.*, vol. 10, no. 12, p. 1841–1844, aug 2017. [Online]. Available: https://doi.org/10.14778/3137765.3137789

[5] L. A. M. C. Carvalho, R. Wang, Y. Gil, and D. Garijo, "Niw: Converting notebooks into workflows to capture dataflow and provenance," in *K-CAP Workshops*, 2017.

[6] S. Samuel and B. König-Ries, "Provbook: Provenance-based semantic enrichment of interactive notebooks for reproducibility," 10 2018.

[7] D. Kerzel, S. Samuel, and B. König-Ries, "Towards tracking provenance from machine learning notebooks," 01 2021, pp. 274–281.

[8] A. Jain, H. Patel, L. Nagalapatti, N. Gupta, S. Mehta, S. Guttula, S. Mujumdar, S. Afzal, R. Sharma Mittal, and V. Munigala, "Overview and importance of data quality for machine learning tasks," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020, pp. 3561–3562.

[9] Y. Shirai, W. Nichols, and M. Kasunic, "Initial evaluation of data quality in a tsp software engineering project data repository," in *Proceedings of the 2014 international conference on software and system process*, 2014, pp. 25–29.

[10] M. Shepperd, "Data quality: Cinderella at the software metrics ball?" in *Proceedings of the 2nd International Workshop on Emerging Trends in Software Metrics*, 2011, pp. 1–4.

[11] K. Sha and S. Zeadally, "Data quality challenges in cyber-physical systems," *Journal of Data and Information Quality (JDIQ)*, vol. 6, no. 2-3, pp. 1–4, 2015.

[12] J. McNaull, J. C. Augusto, M. Mulvenna, and P. McCullagh, "Data and information quality issues in ambient assisted living systems," *Journal of Data and Information Quality (JDIQ)*, vol. 4, no. 1, pp. 1–15, 2012.

[13] S. A. Sheppard and L. Terveen, "Quality is a verb: The operationalization of data quality in a citizen science community," in *Proceedings of the 7th International Symposium on wikis and open Collaboration*, 2011, pp. 29–38.

[14] L. Cao and H. Zhu, "Normal accidents: data quality problems in erp-enabled manufacturing," *Journal of Data and Information Quality (JDIQ)*, vol. 4, no. 3, pp. 1–26, 2013.

[15] H. Xu, "What are the most important factors for accounting information quality and their impact on ais data quality outcomes?" *Journal of Data and Information Quality (JDIQ)*, vol. 5, no. 4, pp. 1–22, 2015.

[16] O. Curé, "Improving the data quality of drug databases using conditional dependencies and ontologies," *Journal of Data and Information Quality (JDIQ)*, vol. 4, no. 1, pp. 1–21, 2012.

[17] P. Barnaghi, M. Bermudez-Edo, and R. Tönjes, "Challenges for quality of data in smart cities," *Journal of Data and Information Quality (JDIQ)*, vol. 6, no. 2-3, pp. 1–4, 2015.

[18] A. Klein, "Incorporating quality aspects in sensor data streams," in *Proceedings of the ACM first Ph. D. workshop in CIKM*, 2007, pp. 77–84.

[19] D. Kontokostas, P. Westphal, S. Auer, S. Hellmann, J. Lehmann, R. Cornelissen, and A. Zaveri, "Test-driven evaluation of linked data quality," in *Proceedings of the 23rd international conference on World Wide Web*, 2014, pp. 747–758.

[20] S. K. Bansal and S. Kagemann, "Integrating big data: A semantic extract-transform-load framework," *Computer*, vol. 48, no. 3, pp. 42–50, 2015.

[21] N. Martin, A. Poulovassilis, and J. Wang, "A methodology and architecture embedding quality assessment in data integration," *Journal of Data and Information Quality (JDIQ)*, vol. 4, no. 4, pp. 1–40, 2014.

[22] K.-S. Na, D.-K. Baik, and P.-K. Kim, "A practical approach for modeling the quality of multimedia data," in *Proceedings of the ninth ACM international conference on Multimedia*, 2001, pp. 516–518.

[23] A. Na'im, D. Crawl, M. Indrawan, I. Altintas, and S. Sun, "Monitoring data quality in kepler," in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, 2010, pp. 560–564.

[24] H. M. Sneed and R. Majnar, "A process for assessing data quality," in *Proceedings of the 8th international workshop on Software quality*, 2011, pp. 50–57.

[25] V. N. Gudivada, R. Baeza-Yates, and V. V. Raghavan, "Big data: Promises and problems," *Computer*, vol. 48, no. 03, pp. 20–23, 2015.

[26] Y. Qin, Q. Z. Sheng, N. J. Falkner, S. Dustdar, H. Wang, and A. V. Vasilakos, "When things matter: A survey on data-centric internet of things," *Journal of Network and Computer Applications*, vol. 64, pp. 137–153, 2016.

[27] V. Gudivada, A. Apon, and J. Ding, "Data quality considerations for big data and machine learning: Going beyond data cleaning and transformations," *International Journal on Advances in Software*, vol. 10, no. 1, pp. 1–20, 2017.

[28] C. Batini, C. Cappiello, C. Francalanci, and A. Maurino, "Methodologies for data quality assessment and improvement," *ACM computing surveys (CSUR)*, vol. 41, no. 3, pp. 1–52, 2009.

[29] V. Ganti and A. D. Sarma, "Data cleaning: A practical perspective," *Synthesis Lectures on Data Management*, vol. 5, no. 3, pp. 1–85, 2013.

[30] D. Loshin, *The practitioner's guide to data quality improvement*. Elsevier, 2010.

[31] C. J. S. R. O. Leo Breiman, Jerome Friedman, *Classification and Regression Trees (1st ed.)*. Chapman and Hall/CRC, 1984.

[32] J. N. Morgan and J. A. Sonquist, "Problems in the analysis of survey data, and a proposal," *Journal of the American Statistical Association*, vol. 58, no. 302, pp. 415–434, 1963. [Online]. Available: https://www.tandfonline.com/doi/abs/10.1080/01621459.1963.10500855

[33] R. Genuer and J.-M. Poggi, "Arbres CART et Forêts aléatoires,Importance et sélection de variables," Jan. 2017, working paper or preprint. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01387654

[34] T. Elomaa and M. Kääriäinen, "An analysis of reduced error pruning," *J. Artif. Int. Res.*, vol. 15, no. 1, p. 163–187, sep 2001.

[35] J. QUINLAN, "Simplifying decision trees," *International Journal of Human-Computer Studies*, vol. 51, no. 2, pp. 497–510, 1999. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1071581987603216

[36] M. J. Berry and G. S. Linoff, *Data mining techniques: for marketing, sales, and customer relationship management*. John Wiley & Sons, 2004.

[37] T. Niblett and I. Bratko, "Learning decision rules in noisy domains," in *Proceedings of Expert Systems '86, The 6Th Annual Technical Conference on Research and Development in Expert Systems III*. USA: Cambridge University Press, 1987, p. 25–34.

[38] J. Mingers, "Expert systems—rule induction with statistical data," *Journal of the operational research society*, vol. 38, no. 1, pp. 39–47, 1987.

[39] L. A. BRESLOW and D. W. AHA, "Simplifying decision trees: A survey," *The Knowledge Engineering Review*, vol. 12, no. 01, p. 1–40, 1997.

[40] J. Mingers, "An empirical comparison of pruning methods for decision tree induction," *Machine learning*, vol. 4, no. 2, pp. 227–243, 1989.

[41] "House price prediction dataset," https://www.kaggle.com/code/shree1992/predicting-house-price/data.

[42] "Code materials available on github," https://github.com/smachar/MLDM_M1_Internship_materials.

Appendices

Appendix A

Error detection using only the satisfied leaf node inequalities

Node	Feature	Feature name	Value	Satisfied	Must Satisfy	score	suggested times	Total score
653	**2**	sqft_living	128.432785	[None, 1225.0]	[3145.0, None]	0.040837	1	0.673869
	11	yr_renovated	0.000000	[None, 2003.5]	[1004.5, 2010.5]	0.000000	1	0.673869
	3	sqft_lot	11374.000000	[6660.0, None]	[17967.5, 20896.5]	0.633032	1	0.673869
191	**2**	sqft_living	128.432785	[None, 1225.0]	[1225.0, 2255.0]	0.104843	3	1.033698
	3	sqft_lot	11374.000000	[6660.0, None]	[None, 2603.0]	0.228855	3	1.033698
	7	condition	5.000000	[4.5, None]	[None, 3.5]	0.700000	3	1.033698
318	**2**	sqft_living	128.432785	[None, 1225.0]	[1225.0, 1665.0]	0.104843	1	1.103498
	12	statezip	98168.000000	[98032.5, None]	[98032.5, 98036.0]	0.998655	1	1.103498
	11	yr_renovated	0.000000	[None, 2003.5]	[2009.5, None]	0.000000	1	1.103498
101	**2**	sqft_living	128.432785	[None, 1225.0]	[1160.0, 1225.0]	0.110718	4	1.370719
	3	sqft_lot	11374.000000	[6660.0, None]	[6660.0, 7222.5]	0.635001	4	1.370719
	1	bathrooms	2.000000	[1.25, None]	[None, 1.25]	0.625000	4	1.370719
52	**2**	sqft_living	128.432785	[None, 1225.0]	[965.0, 1225.0]	0.133091	15	1.718123
	3	sqft_lot	11374.000000	[6660.0, None]	[None, 6660.0]	0.585546	15	1.718123
	12	statezip	98168.000000	[98032.5, None]	[98106.5, 98117.5]	0.999486	15	1.718123
538	**2**	sqft_living	128.432785	[None, 1225.0]	[2255.0, 3145.0]	0.056955	3	1.868487
	12	statezip	98168.000000	[98032.5, None]	[98054.0, 98073.0]	0.999032	3	1.868487
	1	bathrooms	2.000000	[1.25, None]	[None, 1.625]	0.812500	3	1.868487
83	**3**	sqft_lot	11374.000000	[6660.0, None]	[None, 6660.0]	0.585546	1	2.210235
	12	statezip	98168.000000	[98032.5, None]	[98198.5, None]	0.999689	1	2.210235
	1	bathrooms	2.000000	[1.25, None]	[None, 1.25]	0.625000	1	2.210235
14	**3**	sqft_lot	11374.000000	[6660.0, None]	[None, 6660.0]	0.585546	2	2.285032
	12	statezip	98168.000000	[98032.5, None]	[98097.0, 98117.5]	0.999486	2	2.285032
	7	condition	5.000000	[4.5, None]	[None, 3.5]	0.700000	2	2.285032

Figure A.1: *Three-feature changes list ordered by the Totalscore*

Node	Feature	Feature name	Value	Satisfied	Must Satisfy	score	suggested times	Total score
422	2	sqft_living	128.432785	[None, 1225.0]	[2255.0, 3145.0]	0.056955	1	1.709634
	3	sqft_lot	11374.000000	[6660.0, None]	[None, 7428.5]	0.653112	1	1.709634
	12	statezip	98168.000000	[98032.5, None]	[98120.5, 98125.5]	0.999567	1	1.709634
	11	yr_renovated	0.000000	[None, 2003.5]	[1998.5, None]	0.000000	1	1.709634
240	2	sqft_living	128.432785	[None, 1225.0]	[1225.0, 2255.0]	0.104843	5	2.544433
	3	sqft_lot	11374.000000	[6660.0, None]	[2603.0, 6153.0]	0.540971	5	2.544433
	12	statezip	98168.000000	[98032.5, None]	[98029.5, 98032.5]	0.998620	5	2.544433
	7	condition	5.000000	[4.5, None]	[None, 4.5]	0.900000	5	2.544433
403	2	sqft_living	128.432785	[None, 1225.0]	[2255.0, 3145.0]	0.056955	1	2.878493
	3	sqft_lot	11374.000000	[6660.0, None]	[11788.0, 55837.0]	0.964880	1	2.878493
	12	statezip	98168.000000	[98032.5, None]	[None, 98120.5]	0.999516	1	2.878493
	4	floors	1.500000	[1.25, None]	[1.75, None]	0.857143	1	2.878493

Figure A.2: *Four-feature changes list ordered by the Totalscore*