Plaster: An Integration, Benchmark, and Development Framework for Metadata Normalization Methods

Jason Koh*
University of California, San Diego
jbkoh@eng.ucsd.edu

Dezhi Hong*
University of Virginia
hong@virginia.edu

Rajesh Gupta
University of California, San Diego
gupta@eng.ucsd.edu

Kamin Whitehouse
University of Virginia
whitehouse@virginia.edu

Hongning Wang
University of Virginia
hw5x@virginia.edu

Yuvraj Agarwal
Carnegie Mellon University
yuvraj@cs.cmu.edu

ABSTRACT

The recent advances in the automation of metadata normalization and the invention of a unified schema — Brick — alleviate the metadata normalization challenge for deploying portable applications across buildings. Yet, the lack of compatibility between existing metadata normalization methods precludes the possibility of comparing and combining them. While generic machine learning (ML) frameworks, such as MLJAR and OpenML, provide versatile interfaces for standard ML problems, they cannot easily accommodate the metadata normalization tasks for buildings due to the heterogeneity in the inference scope, type of data required as input, evaluation metric, and the building-specific human-in-the-loop learning procedure.

We propose Plaster, an open and modular framework that incorporates existing advances in building metadata normalization. It provides unified programming interfaces for various types of learning methods for metadata normalization and defines standardized data models for building metadata and timeseries data. Thus, it enables the integration of different methods via a workflow, benchmarking of different methods via unified interfaces, and rapid prototyping of new algorithms. With Plaster, we 1) show three examples of the workflow integration, delivering better performance than individual algorithms, 2) benchmark/analyze five algorithms over five common buildings, and 3) exemplify the process of developing a new algorithm involving time series features. We believe Plaster will facilitate the development of new algorithms and expedite the adoption of standard metadata schema such as Brick, in order to enable seamless smart building applications in the future.

CCS CONCEPTS

• **Information systems** → **Entity resolution**; • **Computer systems organization** → *Sensors and actuators*;

KEYWORDS

smart buildings, metadata, machine learning, benchmark

ACM Reference Format:
Jason Koh, Dezhi Hong, Rajesh Gupta, Kamin Whitehouse, Hongning Wang, and Yuvraj Agarwal. 2018. Plaster: An Integration, Benchmark, and Development Framework for Metadata Normalization Methods. In *The 5th ACM International Conference on Systems for Built Environments (BuildSys '18), November 7–8, 2018, Shenzhen, China.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3276774.3276794

1 INTRODUCTION

Smart Buildings have been a major preoccupation of researchers to optimize the energy usage and improve the occupants' comfort as well as the productivity in buildings [43]. Despite the promise over a decade, instrumentation of buildings lags significantly behind, adopted in far less than 20% of the buildings nationwide [2]. This is because of the challenge of connecting sensor data to its semantic context [31]. Smart building applications, such as thermal comfort optimization, fault detection and diagnosis, and model predictive control [13, 36, 37], typically connect to and pull data from the points[1] in a building in order to monitor and access the operations of the building. For example, an application involving a terminal HVAC unit in a room needs to locate the room and its associated points such as on/off commands for the VAV. However, the contextual information of points (e.g., what they measure and how they are related to each other) required by applications to fetch and interpret the data is often lacking — the metadata of point is historically designed mainly for handcrafted control loops, not to be machine-parsed or directly consumed by external software; and the metadata convention varies across sites, if one even exists. To fully realize the potential of smart building applications, a system would need to be able to quickly discover the points and interpret their data in a building in a standardized and uniform way. Doing so would require a common metadata schema for buildings.

Typically, a building metadata schema defines a structure for representing the resources in the building. The representation would comprise two kinds of information about each point in the building — its type and relationships with others. An example is *a temperature sensor is in room 501*, which contains a first entity with the type being temperature measurement, a second entity with the type being room, and the relation between these two entities, i.e., A `is in` B. In the same spirit, Brick, a recently proposed schema

*Co-primary authors

BuildSys '18, November 7–8, 2018, Shenzhen, China

ACM ISBN 978-1-4503-5951-1/18/11.
https://doi.org/10.1145/3276774.3276794

[1] a sensing or control point in the building is a sensor measurement, a controller, or a software value such as a setpoint or a command.

by the research community, is designed to improve over existing industrial building metadata schemata (e.g., Haystack [4], IFC [30], and SAREF ontology [20]) with better expressibility, extensibility, and usability [11, 12]. Brick particularly provides a full hierarchy of entity classes as TagSets and a systematic way of describing various relationships among entities. Brick enables portable building applications built upon common vocabularies of classes and relationships to find required entities, instead of adapting to each individual target building's convention. To instantiate buildings following a schema such as Brick, people commonly rely on parsing the existing metadata in buildings, which can be acquired from the building networks and management systems. Converting such existing metadata into a known structure according to a schema is called *metadata normalization*. However, the normalization process currently requires tremendous manual effort, and for a typical five-story office building with thousands of points and tens of thousands of relationships among them, it can take weeks [21]. An expert with necessary knowledge about the building naming convention and the target standard needs to manually inspect each point out of thousands to correctly map them. This process is clearly not scalable to the millions of buildings, and we need a more usable solution for non-technical users such as building managers to close the loop.

Recognizing this challenge, prior works have proposed methods to partially automate metadata normalization [14, 18, 24, 27, 28, 33, 34, 40, 43], with each focusing on different aspects of metadata. Some methods recognize all entities using the raw metadata [18, 34, 43], including the site, floor and room identifiers, and point type. Other methods identify only the point types based on either the raw metadata [14, 28], timeseries data [24], or both [27]. Yet other methods focus on inferring the relations between entities, including the spatial relationships [26, 32] and functional relationships [33, 40]. In order to reduce the manual effort, these methods either only exploit the structure available within each individual building [14, 18, 24, 28, 43] or transfer information from one building to the next [27, 34]. Importantly, while all of these prior works exploit common attributes of each point — the alphanumeric text-based metadata and/or the numerical time series readings, they differ significantly with regard to the inference scope, input/output format and structure, algorithm interface, evaluation metric, etc [48]. The resultant lack of compatibility among the methods precludes the possibility of combining and comparing them systematically. There is still no standalone, versatile solution so far.

Generic machine learning platforms such as MLJAR [6], OpenML [46], and MLlib [38] have recently emerged. However, while these ML platforms provide generic interfaces for standard machine learning tasks, they are too generic to serve as a usable interface for the unique building-specific human-in-the-loop process with diverse data sources and different input/output formats. We need a modular framework that provides unified interfaces for exploring existing techniques as well as rapidly prototyping new algorithms, in order to advance the state-of-the-art in building metadata normalization. To this end, we design and implement Plaster, a modular framework akin to Scikit-learn for building metadata normalization, which incorporates existing metadata normalization methods, along with a set of data models, evaluation metrics, and canonical functionalities commonly found in the literature. Altogether these enable the integration of different methods into a generic workflow as well as development and evaluation of algorithms. With the designed interfaces, Plaster can easily fit into existing building stacks, from commercial building management systems to open-sourced systems such as XBOS [8] and BuildingDepot [7], that expose the access to metadata and timeseries data in buildings.

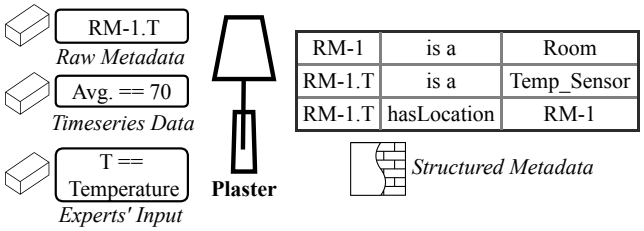


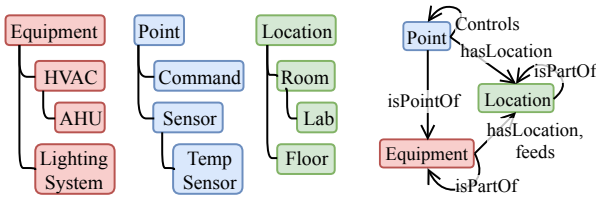
Figure 1: To facilitate portable smart building applications, Plaster collects the state-of-the-art metadata normalization methods and provides a standardized way for users to map unstructured metadata to the Brick format. Plaster also provides a standard benchmark for comparing different methods and spurs new ones.

With Plaster, we also present the first systematic evaluation of the state-of-the-art metadata normalization methods via a set of unified metrics and datasets. Our evaluation covers a wide spectrum of aspects, such as how accurate each method is in inferring the same kind of label, how many different kinds of labels each method can produce, and how many human labels are required to achieve certain performance. The experiment results reveal that there is no one-size-fits-all solution and properly combining them would produce better results. This evaluation would not have been possible without Plaster, given the heterogeneity in earlier works. We believe Plaster provides a comprehensive framework for further development of new algorithms, techniques for metadata normalization, as well as mapping buildings to a structured ontology like Brick, enabling seamless smart buildings applications.

2 BACKGROUND AND RELATED WORK

2.1 Building Metadata Schema: Brick

Without metadata represented in a unified, standardized building-agnostic schema, deploying a smart building application requires adapting it to each target building's naming convention. Thus, the existence and adoption of a standardized metadata schema directly affect the cost of deploying smart building applications [31]. Indeed, there already exist several metadata schemata such as Industry Foundation Classes (IFC) [30] and Project Haystack [4]. However, as they have incomplete vocabularies and cannot fully describe the relationships required by common building applications [17], Brick has been introduced as a complete, extensible, flexible, and usable metadata schema for application portability [11, 12]. Brick comprises a full hierarchy of classes (Fig. 2a) and covers a canonical set of relationships between entities (Fig. 2b). The classes in Brick are also referred to as TagSets as they consist of multiple Tags. For example, `Temperature` and `Sensor` are Tags constituting a TagSet, `Temperature Sensor`. With Brick, one can instantiate the classes to represent actual entities (e.g., a sensor or a room) and relate an entity to another via a particular relationship. The table in Fig. 1 presents an example of a temperature sensor using Brick:



(a) Brick Class Hierarchy (b) Brick Relationships

Figure 2: Brick comprises (a) a full hierarchy of classes and covers (b) a canonical set of relationships between entities required by common smart building applications.

the original raw metadata `RM-1.T` is mapped to an instance of `Temperature Sensor`; and to represent relational information such as its location, one can explicitly associate it with other entities such as room-1, which is again an instance of type `Room`. With Brick, a user can avoid using custom tags to describe both the entity type and its relationships with others, which makes running portable applications across buildings feasible. Therefore, we choose Brick as the target mapping convention in this paper as it is capable of representing the resources and relationships needed in smart buildings, and is in our opinion more comprehensive than other schemata.

2.2 Metadata Normalization Methodologies

We identify three dimensions of variance in existing metadata normalization methods: 1) the type of data sources exploited, 2) the kinds of labels produced, and 3) the degree of human input required.

There are three different types of **data sources** we can exploit in buildings. The raw **metadata** in BMSes, also referred to as *point names*, usually encodes various kinds of information about the control and sensing points, including the type of sensor, floor and room numbers, HVAC equipment ID, etc. The metadata within a building often exhibits a strong learnable pattern, though it varies significantly across buildings and often does not generalize from one building to another, and various works have leveraged such pattern for metadata inference [14, 18, 28, 34, 43]. Secondly, modern BMSes also collect **time series** readings of each point in the building, which contain information that indirectly reveals what the point is and its relationship with others. For example, the range of the readings can indicate the type of sensor and the correlated changes in different streams can indicate the relationship. Works that leverage the characteristics of timeseries data include [24, 26, 32]. Additionally, one may also perform **controlled perturbation** in a building, e.g., to manually turn off an air handling unit, and create new patterns in operations that help to reveal the functional relationships between entities more clearly [33, 40]. However, it requires careful and sophisticated designs with regard to the system configurations and inhabitants' schedules.

Existing metadata normalization methods focus on producing two **kinds of labels** — following the definitions in Brick — entity types and relationships between entities. The entity type refers to the type of measurement of a point and there is a wide variety in its possible set of labels, while the relationships include how points are connected to each other, whether they are in the same room/zone, etc. A few methods infer all the available information (e.g., both kinds of labels) encoded in the raw metadata [18, 34, 43], whereas many others identify the point type only [22, 24, 27, 28], which is the most important aspect of a point in buildings, or infer the relationships only [26, 32, 33, 40].

Table 1: State-of-the-art metadata normalization methods produce various types of labels using different data sources. They also employ different machine learning (ML) algorithms involving diverse types of user interaction.

Method	Label Produced	Data Source	ML
Bhattacharya et al. [18] Scrabble [34]	All entities	Raw metadata	AL
Zodiac [14] Hong et al. [28]	Point type	Raw metadata	AL
Fürst et al. [22]	Point type	Raw metadata	CS
BuildingAdapater [27]	Point type	Raw metadata, Timeseries	TL
Gao et al.[24]	Point type	Timeseries	SL
Hong et al.[25]	Point type	Timeseries	UL
Pritoni et al. [40] Quiver [33]	Functional Relationship	System Perturbation, Point Label	UL

AL: Active Learning TL: Transfer Learning
SL: Supervised Learning CS: Crowd Sourcing
UL: Unsupervised Learning

While different methods all aim to reduce the amount of manual effort in normalizing metadata, the degree of **human input** required by each of them varies from fully supervised to semi-supervised to completely unsupervised. Particularly, supervision, or human input, in this context is the annotation or labels that a human expert provides to interpret the point for its type, location, relationship with others, etc. Supervised learning has been used to learn the point types based on timeseries data or raw metadata, where both clean, accurate labels from experts [24] and crowdsourced labels from occupants in the building [22] have been explored in the literature. For the set of semi-supervised solutions, they employ active learning to iteratively select the most informative example and query an expert for its label to improve a model for normalizing the metadata, requiring the minimal amount of labels [14, 18, 28, 34]. On the other hand, transfer learning method has been developed to exploit information from existing buildings and completely eliminate human effort when inferring the metadata in a target building [27]. Similarly, Scrabble [34] is another method that exploits existing buildings' normalized metadata, but through an active learning procedure. Table 1 summarizes these various methods with regard to the above criteria. In this work we show that, while each of these techniques has its advantages, our proposed *meta*-framework – Plaster– can help to choose the right algorithm per user requirement as well as leverage different techniques in a complementary manner to yield better results.

Generic machine learning platforms such as MLJAR [6], OpenML [46], Microsoft Azure ML Studio [5], and MLlib [38] have recently emerged. These platforms have proved to be useful and facilitated tasks and research on machine learning. However, a metadata normalization problem has more unique requirements: 1) it handles diverse types of input/output data, receiving as input timeseries data and/or encoded textual metadata, and produces a graph (such as Brick entity graph) as a final output, 2) it involves various types of learning

framework including transfer learning, active learning, and supervised learning altogether, and 3) users would need to interact with the algorithm(s) through the abstraction of the building data, rather than directly with the data itself. Consequently, existing frameworks cannot be adopted for metadata normalization tasks. Additionally, although not being directly related to the metadata normalization problem, there are frameworks in other domains that integrate different algorithms and create composable workflows, including general machine learning analytics [10], recommendation systems [29, 49], non-intrusive load monitoring [15, 16]. To the best of our knowledge, Plaster is the first framework of its kind that enables the exploration and integration of various algorithms on building metadata normalization, as well as provides the ability to systematically compare related algorithms.

3 PLASTER FRAMEWORK

Plaster delivers a modular framework for benchmarking, integration, and development by providing two levels of abstractions common among existing methods. As the *first* level of abstraction, Plaster views a metadata normalization task as an ensemble of a key **inferencer** and several other reusable components that have canonical functionalities and interfaces. This way, we provide users with the flexibility in choosing the data model, learning scope, and inference algorithm as needed. As the *second* level of abstraction, an inferencer, which is the core component, comprises multiple common functions that we identify by summarizing existing metadata normalization solutions. Because of the unified interfaces and its modular design, Plaster facilitates the invention of new *workflows* where a user can connect different inferencers to essentially create a new algorithm without re-implementing prior algorithms.

3.1 Architecture

In Plaster, we abstract each method as an ensemble of components, and overall there are four categories of components as illustrated in Fig. 3a: preprocessing, feature engineering, inference models, and results delivery functions.

The *preprocessing* component includes standard functions such as denoising and outlier removal for timeseries data, and lowercasing and punctuation removal for textual metadata, via an interface to utilize existing libraries such as SciPy and Pandas. There are also database (DB) I/O functions for both the metadata and timeseries data. We use universally unique identifiers to identify points and one can access both the textual metadata and timeseries data through the identifiers. For the timeseries DB functions, Plaster builds upon an open-source library [3] piggybacked on MongoDB, which is dedicated and optimized for timeseries data operations on large data chunks. For *feature engineering*, there are a number of existing libraries, such as the most widely used scikit-learn [39] and a recent effort – tsfresh [19]. However, none exists as customized for the timeseries data from buildings, considering their uniqueness such as the distinct diurnal patterns. Hence, we incorporate and extend the feature sets[2] implemented by Gao et al. [23], which contain various feature functions customized for building timeseries data. In addition to the original feature sets, we provide straightforward programming interfaces for a user to select a subset of features out of these predefined features and a lightweight yet effective feature selection function based on lasso [45]. We shall demonstrate the effectiveness of the feature selector in Section 4.4. For text features, we provide an interface for Bag-of-Words [42], sentence2vec [35], and LSTM-based auto-encoder [44]. *Delivery* components consist of the set of evaluation metrics (detailed in Section 4.1.2), user interaction mechanisms to provide additional supervision for inferencer update as needed, and serialization tools that convert the inference results into the Brick format (e.g, triples and graphs).

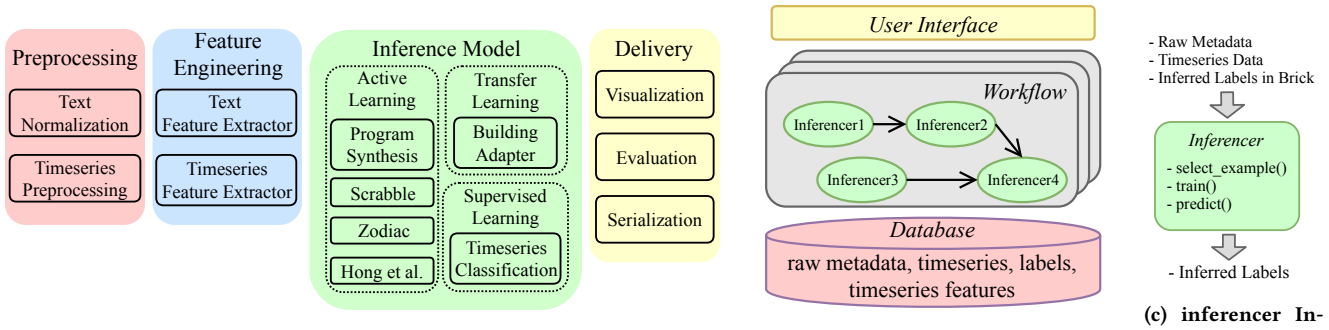
[2]We refer the readers to their original paper [23] for more details on each feature set.

3.2 Inferencer

At the core of Plaster is a collection of state-of-the-art metadata inference methods. We examine these algorithms and identify similar procedures among them. We therefore abstract these procedures as a series of common functions, encapsulate each as a parameterized interface, and formulate a standardized way of constructing an inference algorithm. We use an abstract class – inferencer– to represent an algorithm (e.g., Scrabble, Building Adapter, etc), which maintains its own model for metadata normalization under these abstract interfaces. Such abstraction decouples the complex procedures in individual algorithm and allows new algorithms to be easily included into the framework.

At a high level, an inference algorithm in the building metadata domain aims to achieve the best possible accuracy with the largest coverage using the minimal set of labeled examples. Therefore, an inferencer typically contains a few steps: 1) the algorithm selects as training set the most "informative" example(s) based on its own criterion and acquires the labels for the selected example(s) from a human expert; 2) the model updates its parameters based on the latest training set after the new examples are added in the previous step, and then 3) the model predicts all types of labels (e.g., point type, location, relationships, etc.) covered by the algorithm. Plaster abstracts each of the above steps as a function, viz, `select_examples()`, `train()`, and `predict()`, respectively, as shown in Fig. 3c; and we design an inferencer to be a composition of these functions. We shall note that, although these functions appear to be only able to compose an active learning-based procedure, we design the `select_examples()` function to be generic enough such that any fully to semi-supervised learning algorithm can fit into this template. When obtaining examples for a supervised or transfer learning algorithm, the `select_examples()` function simply includes all the labeled or transferred examples for training at one time, rather than being iteratively done as in active learning. For active learning, these steps are repeated in iterations involving a human expert to best learn the model, while for a supervised learning or transfer learning algorithm, these steps are mostly executed just once with already labeled examples.

We also define standardized input/output interfaces for these common functions to enable the communication between different inferencers, which permits the creation of workflow as we shall discuss shortly. For inputs, an inferencer accepts three types of sources: raw metadata, timeseries data, and the corresponding labels of examples. We provide a wrapper to digest two types of raw metadata commonly found in existing systems: 1) point names accessible through vendor-given interfaces that are widely used in the literature, and 2) metadata in BACnet [9] including entries



(a) Components in Plaster **(b) Workflow** **(c) inferencer Interface**

Figure 3: Plaster Architecture: Plaster adopts a modular design and incorporates a variety of components, among which the core is a family of inference algorithms. Each algorithm is abstracted as an ensemble of common functions, which allows the communication between different algorithms. Such a design enables not only the flexible invention of a workflow composed of any algorithms, but also the systematical comparison between different algorithms.

such as `BACnet Description` and `BACnet Unit`. Timeseries data is stored as a series of timestamped values and the data for each specific point is associated with a unique identifier of the point for indexing and future retrieving. As each inferencer can learn and produce various types of labels as discussed in Section 2.2, an inferencer can take three kinds of labels at different granularities: point type labels, labels for all entities existing in the metadata, and character-level parsing with BIO tagging [41].

The ultimate goal of each individual inferencer is to generate structured metadata. Since Brick is capable of representing different kinds of metadata such as the types of entities and the relationships between them, we express the `predict()` method's outputs of each inferencer following the Brick's format. Particularly, the outputs are a list of triples for entities and relationships in a building as explained in Section 2.1. Consequently, an inferencer is capable of representing different inference results in the same format. For example, Zodiac [14] infers the point types, which can be represented as "X is a Y" triples, while Quiver [33] infers the co-location relationship for multiple sensors expressed as "X1 hasLocation Y" and "X2 hasLocation Y". Such different types of inference are serialized in the same format of Turtle [1] using the vocabulary in Brick.

Additionally, for each inferencer we include the confidence of its inference results produced by the original algorithm in its output so that an inferencer is able to more flexibly sift through and use another inferencer's results. Specifically, we store the confidence for each produced triple within the inferencer. However, the notion of confidence is unique per inference algorithm with different meanings. For example, Support Vector Machine's confidence is usually measured by the distance to the hyperplane, while in Naïve Bayes, it is the probability of observing the example given the model's parameters. Thus, we restrict the interpretation of confidence score within each individual inferencer despite the values of those metrics being uniformly normalized to be between zero and one. We shall show how this is useful in real workflows in Section 4.3.

As a natural outcome, Plaster provides a standard benchmark for different metadata normalization algorithms. With the unified interfaces in inferencer, to do so is straightforward as one only needs to specify the set of algorithms he/she wants to compare as well as the type(s) of data to ingest and designate a building for the comparison. We will demonstrate with concrete examples in Section 4.2.

3.3 Workflow

The standardized interfaces in inferencer also enable the creation of a workflow for metadata normalization. A workflow is a hybrid method comprised of multiple algorithms, each being an inferencer in Plaster, where the output of an inferencer is passed to another while each inferencer executes its inference procedure independently. While a single inferencer usually only infers one aspect of the metadata, a workflow would potentially be able to infer multiple or all the aspects of the metadata by employing different inferencers. Each inferencer may have a different learning objective as described in Section 2.2, and Plaster helps to systematically leverage the advantages of each. For example, Building Adapter [27] (BA) is a transfer learning based algorithm that infers point types without any human inputs, but usually with a potential low recall. Instead of starting from scratch, the output of BA can constitute an initial training set for Zodiac [14] to jump start its learning procedure and potentially reduce the amount of manual labels required. Various use cases of workflow enabled by Plaster are elaborated and evaluated in Section 4.3.

When executing, the workflow function call will invoke the corresponding functions (i.e., `select_examples()`, `train()`, and `predict()`) in each of its inferencers in the order specified in the workflow, with an additional connecting step that obtains and applies the previous inferencer's prediction results to the next. The process of applying a precedent inferencer's results vary across different inferencers so that a human integrator should specify how to digest such predictions inside the inferencer's methods. If some of the inferred relationships by the previous inferencer are less confident, the next inferencer should filter the results or simply avoid using them. On the contrary, if the previous inferencer's inference is more confident than the current inferencer's, it can discard its own inference and adopt the previous one. In the example of connecting BA and Zodiac, Zodiac would need to be able to select only the prediction results with high confidence from BA and subsequently

add them into its own training set inside `select_examples()`. Although it is an additional implementation over the base algorithms, we shall see such synergy could lead to better results.

4 EVALUATION

In this section, we demonstrate how Plaster enables systematic comparisons of different metadata normalization algorithms, the creation of new workflows by connecting multiple algorithms, and the programming interfaces for algorithm development such as feature selection.

4.1 Experimental Setup

4.1.1 Datasets. We obtain a subset of the study buildings used in Brick [11], which consists of five buildings from four different campuses, including the raw metadata and timeseries data for about a month. Table 2 summarizes the details of each test building. While this collection of five buildings are not comprehensive for building metadata research, we argue that they are representative enough with regard to the diversity in vendors, sizes, years of construction, etc. For building D1, the original author did not release the timeseries data, and therefore, we shall note that D1 will not be included later in evaluations that involve timeseries data.

4.1.2 Evaluation Metrics. Overall, we consider three aspects when evaluating each algorithm:

- *Inference Accuracy*: How accurate are the predictions of an algorithm in terms of its original learning purpose?
- *Inference Coverage*: What kinds of labels can an algorithm infer?
- *Human Efforts*: How many examples does an expert need to provide in the learning process of an algorithm?

In this study, each algorithm infers one or multiple kinds of labels for a point. For example, Zodiac infers only one kind of labels, which is the point type, whereas Scrabble also identifies other kinds of labels such as location aside from the point type. For each kind of label, every possible Brick tagset is treated as a class (e.g., for point type we have room temperature, supply air temperature, etc), and we evaluate the inference performance considering all kind(s) of labels each algorithm produces. To measure how accurate the inference results are for an algorithm, we calculate the Micro-averaged F1 (MicroF1), Macro-averaged F1 (MacroF1), and example-level accuracy. MicroF1 globally counts the total true positives, false negatives and false positives regardless of the class, while MacroF1 calculates the same quantities for each class and then finds their unweighted mean. MacroF1 indicates how many different classes can be correctly inferred, which is an important metric for a building dataset that typically has an (extremely) imbalanced class distribution, with the points related to heating and cooling in domination. For example, while `Zone Temperature Sensors` might frequently exist in HVAC systems, specialized points such as `Gas Meters` are generally rare. For example-level accuracy, it is defined as the ratio of the number of correctly labeled examples over the total number of examples. Specifically, an examples is considered to be correctly labeled if and only all of its labels are correctly predicted. We use this metric along with the F1 scores when an algorithm produces more than one kind of labels.

We measure human efforts by the number of examples labeled by an expert during the model learning process. For point type inference, an example is usually a mere point type label given the raw metadata of the point. For the examples used for inferring all possible entities, they contain more information aside from the point type label, such as equipment ID and location. Although the amount of information in the examples are different, we consider the effort for labeling an example to be the same because the required knowledge per example is similar.

4.1.3 Inferencers Included in Plaster. We have refactored and incorporated the following algorithms into Plaster: Hong active learning [28] (referred to as AL_Hong hereafter), Bhattacharya et al. [18] (referred to as ProgSyn), Zodiac [14], Building Adapter [27], and Scrabble [34]. We exclude algorithms from the evaluation that require the actual control of actuations in buildings [33, 40] because such experiments are not practical in most buildings. However, they fit into Plaster well as part of a workflow in the real world such as building commissioning. Plaster is open-sourced and implemented in Python. The API documentation, running examples, together with the data sets can be found at
https://github.com/plastering/plastering.

4.2 Benchmarking

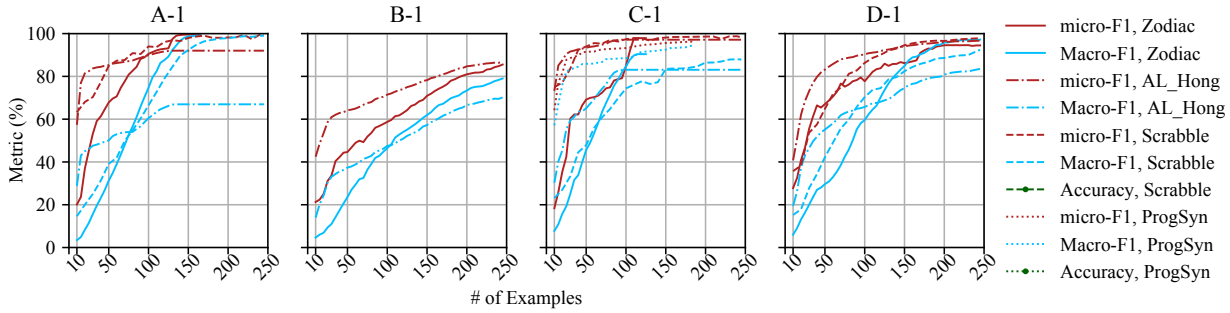
Enabled by the unified interfaces in inferencer, Plaster allows a user to easily select a method and specify the type of input ingested, the test building to use, and the evaluation metric; this facilitates systematical comparisons of different algorithms, i.e., *benchmarking*. We present the results of three representative scenarios.

4.2.1 Active Learning for Point Type Inference. In this scenario, we evaluate a set of active learning-based algorithms for their learning efficiency in inferring point types, the most important aspect of building metadata. We include two algorithms that exclusively work for this purpose – AL_Hong [28] and Zodiac [14], together with another two algorithms that can infer multiple aspects in metadata (type, location, equipment, etc) – ProgSyn [18] and Scrabble [34]. Although the latter two are designed to learn all aspects of metadata, we make each to infer only the point type in this set of experiments. We run each algorithm on four different buildings, starting with zero training examples, and calculate the MicroF1 and MacroF1 of inferred type labels. The results are shown in Fig. 4a.

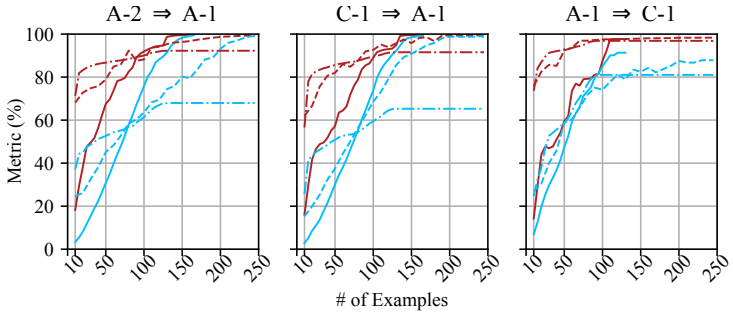
We see that AL_Hong marks a stark contrast to all the other algorithms for its steep learning rate (by MicroF1) in the early stage for the first 75 examples. This is because of its clustering-based example selection strategy, which excels in quickly selecting representative examples that are also informative for model training. However, we also see that Zodiac and Scrabble are able to catch up after 75 to 125 examples, surpassing in MacroF1, and even achieve 100% in F1 for some case (on building A-1) after converging. These results suggest that Zodiac and Scrabble are better in learning the *minor* point types that appear less frequently in a building, which AL_Hong is not able to learn even with more examples. We would also like to point out that, due to the deterministic nature of the algorithm, ProgSyn and Zodiac may terminate early (e.g, on C-1). Zodiac runs with a preset confidence threshold, and as it gradually acquires training examples, whenever the algorithm has

Building	Location	Vendor	Year	Size (ft^2)	# Points	# Point Types	# Unique Words
Engineering Building Unit 3B	UC San Diego, San Diego, CA	JCI	2004	150,000	4,594	108	426
Applied Physics and Mathematics	UC San Diego, San Diego, CA	JCI	2004	150,000	4,357	111	369
Rice Hall	Univ. of Virginia, Charlottesville, VA	Trane	2011	100,000	1,300	60	290
Sutardja Dai Hall	UC Berkeley, Berkeley, CA	JCI	2009	141,000	2,300	31	116
Gates Hillman Center	Carnegie Mellon Univ., Pittsburgh, PA	ALC	2009	217,000	8,292	147	179

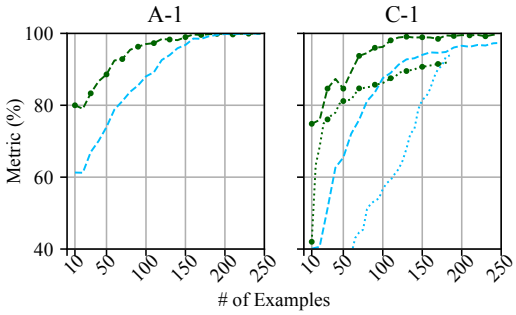
Table 2: Case Study Buildings Information: These are office buildings in universities. JCI and ALC stand for Johnson Controls and Automated Logic, respectively. The number of unique words represents the complexity of the metadata.



(a) Learning rate for inferring point type by different algorithms on 4 buildings starting from scratch (i.e., zero training set).



(b) Learning rate for inferring point type, exploiting an existing building's normalized metadata. X -> Y indicates applying X's normalized metadata to initialize the learning for Y.



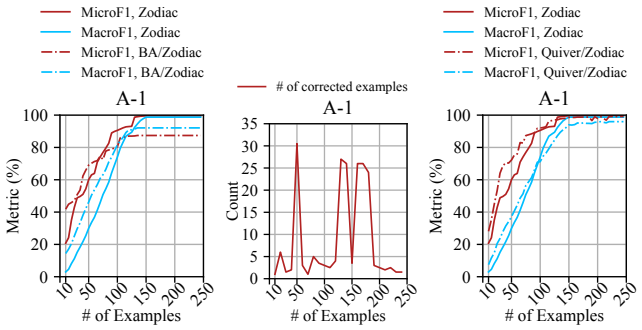
(c) Learning rate for inferring all entities in the raw metadata from scratch.

Figure 4: Comparisons of Different Algorithms on Various Buildings: (atop each figure) The alphabet represents a campus and the number represents a building on that campus (e.g., A-1). We leave out Scrabble's results for B-1 and ProgSyn's results for A-1, B-1 and D-1 due to the limited types of labels in these buildings. All experiments are averaged over four runs and the legend is shared across all figures.

high enough confidence in every testing instance, it will cease. For ProgSyn, it decides whether the learned rules are able to parse every example and stops when it becomes the case. Furthermore, there is no clear winner in this set of experiments. The implication, however, is that if one wants to quickly label the types with reasonably high accuracy (e.g., 85%), AL_Hong is an appropriate choice. When one desires better coverage of less frequent types in a long run, Zodiac or Scrabble would be a better choice.

4.2.2 Jump-started Active Learning. All the original active learning based algorithms [14, 28, 34] are designed to work only within the same building, meaning that they do not consider or leverage any information from other existing buildings. However, because the inferencer design in Plaster makes it convenient to start from any training set, we will next show what the learning results would be if we run an active learning-based algorithm using information from another building for inferencer initialization. More specifically, we use another building's point names along with their labels (e.g., from A-1) to formulate the initial training set for an inferencer and then run the algorithm on another building (e.g., C-1) as we did in Section 4.2.1. The results are shown in Fig. 4b.

When added a building from a different vendor with almost completely distinct naming conventions (e.g., A-1 -> C-1 and C-1 -> A-1), the type inference performance either remains unchanged or even deteriorates in the early stage. This is expected as such transfer would introduce more irrelevant patterns to the same point type for the algorithm to learn, which is almost equivalent to injecting noise. Nonetheless, we still notice an increase in MicroF1 for Scrabble in the early stage in the case of A-1 -> C-1. This is largely due to Scrabble's underlying intermediate representation, which is



(a) BA -> Zodiac **(b) Zodiac -> Scrabble** **(c) Quiver -> Zodiac**

Figure 5: Learning efficiency for inferring point type by different workflows. They all demonstrate synergistic improvement in performance.

able to learn more general patterns with different buildings. On the other hand, when we add a building from the same vendor with a similar vocabulary for point types (see A-2 -> A-1), we observe a better starting point (71% in the first figure in 4b vs 58% in the first figure in 4a) and also a better converged MicroF1 for AL_Hong. We observe similar improvements for Scrabble and Zodiac in this case. We thus conclude that having building(s) with similar naming convention is useful for inferring subsequent buildings by transferring the information in the raw metadata.

4.2.3 Active Learning for Multiple Entities. While detecting the point type is important, other types of entities encoded in the metadata, such as the associated room and equipment, are also essential for building applications. We therefore evaluate Scrabble and ProgSyn for their ability to identify multiple types of entities from the given raw metadata, including the point type, room location, and associated equipment ID. As shown in Fig. 4c, Scrabble outperforms ProgSyn in both MacroF1 and example-level accuracy. The gain in performance of Scrabble is attributed to its more sophisticated representation learning procedure where it first maps the input to an intermediate representation and then to actual labels, while ProgSyn maps the raw metadata directly to final labels. For example, for a string `ZNT`, Scrabble first learns its nuanced character-level BIO tags and then maps to the Brick tagset (i.e.e, `Zone Temperature Sensor`), while ProgSyn directly learns its mapping rule to the tagset via regular expressions.

4.3 Workflow

Having seen the results on comparing different algorithms individually, we next show how they can interact with each other in Plaster. A key feature of Plaster is the ability to try out different workflows, which integrate different algorithms in different orders. We present and evaluate three exemplary workflows where two inferencers are connected together for better performance than if they are used individually.

4.3.1 Transfer Learning Benefits Active Learning. A completely automated method such as Building Adapter (BA) [27] is able to achieve relatively high inference precision for point types in a target building, though for only a fraction of the points. It would be natural to connect and feed the labeled examples by BA to an active learning-based method, such as Zodiac [14], as a better starting point. This appears to be similar to the jump-started active learning scenario in Section 4.2.2, in that both provide a better starting point for active learning. However, a fundamental difference is that a method such as BA, which transfers the learnt model via *timeseries data* from a different building to facilitate another learning process based on *textual data*, which is independent from these two buildings' naming conventions, while in the previous scenario we will only see benefits when transferring from a building with a similar naming convention. We implement such a workflow of combining BA and Zodiac to again infer point types, with Fig. 5a showing the comparison results. We see that the combination achieves both higher MicroF1 and MacroF1 up to 70 examples, benefiting from the transferred information. However, the incorrect labels from BA's predictions (though only a handful) remain as negative training examples to Zodiac and it cannot recover from such noise in such a naïve integration. These inherited incorrect labels would be corrected or filtered out at the beginning if Zodiac could have the ability to quantify BA's results based on its own criterion. Yet, this will require additional modifications to the original algorithm and is hence out of the scope of Plaster.

4.3.2 Specialty Complements Versatility. Some algorithms have high precisions while others have high recalls in their inference results. For example, Zodiac infers only point types but with high precision, while Scrabble can identify multiple kinds of entities with a high recall. Thus we can filter Scrabble's results by using Zodiac's results without compromising the results of either. More specifically, we feed Zodiac's results to Scrabble's prediction and if there is a disparity between the two on an instance, Scrabble will adopts Zodiac's prediction for point type. As shown in Fig. 5b, we see there are about 1,500 corrections made to the point type predictions in total (note that we only count the number of corrections made by this strategy, and an instance could be corrected multiple times) with little additional computational cost.

4.3.3 Mutual Benefits between Different Types of Inference. Learning functional relationships often relies on perturbations to the control systems (e.g., Quiver [33]) and, to correctly perform perturbations on a target point such as a VAV on/off command, it requires knowing the point types apriori. Thus, it is natural to apply an active learning algorithm (Zodiac) to infer point types as a prior step to a perturbation-based relationship inference algorithm (Quiver). Furthermore, the inferred relationships can in return help examine whether the point types have been correctly inferred. For example, the fact that a VAV typically contains only one for each type of its sensing and control points can help identify mistakes made in type inference. Concretely, based on the manual perturbation to a VAV on/off command, Quiver [33] identifies a group of co-located points and finds that there are two supply air temperature sensors; it is highly likely that Zodiac has made a mistake in the type inference. For this experiment, we emulate the above procedure by first running Zodiac to infer point types, and for each predicted VAV on/off command, we use the ground-truth for the co-located points in that VAV (since we are not able to actually run Quiver) and examine if there is any duplicated type among these points,

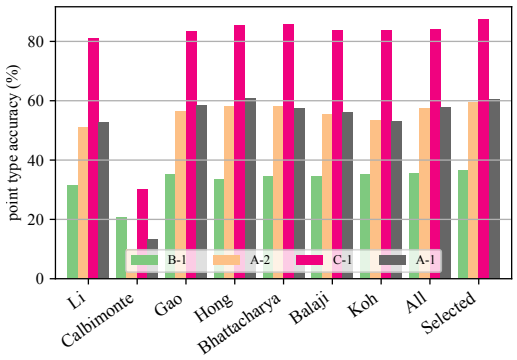


Figure 6: Results for timeseries data based type inference: We show 7 different feature sets (each is shown as a group of bars and each bar represents the result on a building), along with a fusion of them (All) and a better subset selected by Plaster (Selected).

in order to correct any type mis-predictions. We see modest improvement in the type inference results because we only consider ~15 most common point types existing in VAVs, as Quiver can only find the co-located points for VAVs. Although a workflow as such exploits certain domain knowledge, it would be generally useful to practitioners with special demands in building applications.

4.4 Timeseries Feature Selection

We also empirically inspect how well each timeseries feature set performs and how effective the feature selection is in Plaster. To this end, we create a workflow that feeds the timeseries data to each of the feature extraction modules included in Plaster, passes the features to a random forest classifier (which is identified as the best performing classifier [23]), and predicts the point type for evaluation. Fig. 6 summarizes our results.

We observe that each individual feature set roughly performs on a par except the second set. A simple fusion of all the dimensions from each feature set (marked as All in the figure), which equates to a 106-dimensional feature set, does not yield much better performance. However, the selected set of features does give a 3% increase overall than the best set with the number of features reduced to 60. This demonstrates the usefulness of the feature selection and integration provided by Plaster.

We clearly notice the performance here by using timeseries data features is far less competitive than using textual metadata (as demonstrated in Fig. 4). However, the implication is that, as data features better suit transfer learning based tasks [27], the better feature set we have identified here would help to improve such a procedure, for instance Building Adapter. Moreover, we would also like to emphasize that subsequent users and/or researchers can easily register their own feature set in the feature extractor interface in Plaster, and also perform feature selection with our provided method to obtain a even better set of features for their target metadata normalization problem.

4.5 Programming APIs and Examples

We next showcase a simple code snippet on how to evaluate an inferencer in Plaster following the unified interface design. We see from the example that one only needs to specify an algorithm along with some configurations including the buildings involved for evaluation and parameters to the algorithm. We shall note that for more running examples on benchmarking, workflow, etc, one can refer to our documentation[3] for details.

```
target_building, source_buildings = 'ap_m', ['ebu3b']
SAMPLE_SIZE = 100
srcids = LabeledMetadata\
    .objects(building=target_building).distinct('srcid')
training_srcids = random.sample(srcids, SAMPLE_SIZE)
test_srcids = [srcid for srcid in srcids
               if srcid not in training_srcids]
scrabble = ScrabbleInterface(target_building, srcids,
      source_buildings)
scrabble.update_model(training_srcids)
metrics = scrabble.evaluate(test_srcids)
```

Listing 1: Example for Evaluating an Inferencer

5 DISCUSSION AND FUTURE WORK

First, although not being the focus of Plaster, we see that for the same metadata normalization algorithm, still its performance varies significantly from building to building. When designing an algorithm in the future, one would want to test on a diverse set of buildings to avoid over-fitting for a single building. Secondly, having seen the synergy between different algorithms via our workflow design, the original authors and/or subsequent researchers might consider revisiting and refining the design of existing metadata normalization algorithms, in order to better leverage the advantages from each other. Thirdly, given the recent advances in deep neural networks concerning both textual and timeseries data, it remains open how to best harvest their progress to complement or reshape the research on metadata normalization. Plaster now provides some basic functions for feature selection taking advantage of existing neural network algorithms. Furthermore, the architecture of Plaster is flexible enough to directly develop neural network based algorithms. Additionally, as Plaster currently is only compatible with inputs in the Brick format, in future we would want to extend it to exploit existing resources available in other schemata such as Haystack, in order to augment the existing methods in Plaster via a workflow. Furthermore, we would like to provide the ability of automatically selecting an inferencer, or composing a workflow, for a user based on their demands and what are available to them, i.e., the idea of meta-learning [47]. For example, if they want to convert a building with a few others available already, we could connect Building Adapter and Scrabble as a workflow for them.

6 CONCLUSION

Connecting sensor data to the context in which it was generated and mapping this information to a normalized format is challenging. The recent invention of a unified schema – Brick – and various metadata normalization methods alleviates the challenge. Yet, the lack of compatibility between methods precludes the possibility of combining and comparing them due to the heterogeneity in the inference scope, input/output format and structure, algorithm interface, and evaluation metric. In this paper, we present Plaster,

[3] https://github.com/plastering/plastering/blob/master/examples

a modular framework with unified interfaces, which enables the creation of workflows as well as development and evaluation of new algorithms. Via systematic evaluations with a set of unified metrics and building datasets, we have demonstrated that *for the first time*, Plaster provides a standard benchmark for different metadata normalization methods, and the workflows can integrate existing methods. Our results reveal that 1) each method has their own pros and cons and should be chosen according to a user's requirement, and 2) different methods can be combined in a complementary manner to yield better results. We believe Plaster provides a useful framework for further advances in metadata normalization for buildings, as well as for mapping metadata to a schema like Brick, enabling seamless smart buildings applications in the future.

ACKNOWLEDGMENTS

We thank our shepherd, Clayton Miller, and the anonymous reviewers for helpful comments. This work was supported by National Science Foundation [CNS-1526841, CSR-1526237, TWC-1564009, IIS-1636879, IIS-1718216] and Department of Energy [DE-EE0008227].

REFERENCES

[1] [n. d.]. Turtle. https://www.w3.org/TR/turtle/.
[2] 2012. Commercial Buildings Energy Consumption Survey 2012—Microdata. https://www.eia.gov/consumption/commercial/data/2012/.
[3] 2014. Arctic. https://github.com/manahl/arctic. last visited: 05-20-2018.
[4] 2014. Project Haystack. http://project-haystack.org/.
[5] 2016. Microsoft Azure Machine Learning Studio. https://studio.azureml.net/.
[6] 2016. MLJAR. https://mljar.com/.
[7] 2017. BuildingDepot 3.0. http://buildingdepot.org/.
[8] Michael P Andersen, John Kolb, Kaifei Chen, David E Culler, and Randy Katz. 2017. Democratizing authority in the built environment. In *BuildSys*. ACM, 23.
[9] ASHRAE 135-2016 BACnet 2016. *BACnet-A Data Communication Protocol for Building Automation and Control Networks*. Standard. ASHRAE, GA, USA.
[10] Peter Bailis, Kunle Olukotun, Christopher Ré, and Matei Zaharia. 2017. Infrastructure for Usable Machine Learning: The Stanford DAWN Project. *CoRR* abs/1705.07538 (2017). arXiv:1705.07538 http://arxiv.org/abs/1705.07538
[11] Bharathan Balaji, Arka Bhattacharya, Gabriel Fierro, Jingkun Gao, Joshua Gluck, Dezhi Hong, Aslak Johansen, Jason Koh, Joern Ploennigs, Yuvraj Agarwal, et al. 2016. Brick: Towards a unified metadata schema for buildings. In *BuildSys*.
[12] Bharathan Balaji, Arka Bhattacharya, Gabriel Fierro, Jingkun Gao, Joshua Gluck, Dezhi Hong, Aslak Johansen, Jason Koh, Joern Ploennigs, Yuvraj Agarwal, et al. 2018. Brick: Metadata schema for portable smart building applications. *Applied Energy* (2018).
[13] Bharathan Balaji, Jason Koh, Nadir Weibel, and Yuvraj Agarwal. 2016. Genie: a longitudinal study comparing physical and software thermostats in office buildings. In *UbiComp*. ACM, 1200–1211.
[14] Bharathan Balaji, Chetan Verma, Balakrishnan Narayanaswamy, and Yuvraj Agarwal. 2015. Zodiac: Organizing large deployment of sensors to create reusable applications for buildings. In *BuildSys*. ACM, 13–22.
[15] Nipun Batra, Jack Kelly, Oliver Parson, Haimonti Dutta, William Knottenbelt, Alex Rogers, Amarjeet Singh, and Mani Srivastava. 2014. NILMTK: an open source toolkit for non-intrusive load monitoring. In *Proceedings of the 5th international conference on Future energy systems*. ACM, 265–276.
[16] Christian Beckel, Wilhelm Kleiminger, Romano Cicchetti, Thorsten Staake, and Silvia Santini. 2014. The ECO data set and the performance of non-intrusive load monitoring algorithms. In *Proceedings of the 1st ACM Conference on Embedded Systems for Energy-Efficient Buildings*. ACM, 80–89.
[17] Arka Bhattacharya, Joern Ploennigs, and David Culler. 2015. Short paper: Analyzing metadata schemas for buildings: The good, the bad, and the ugly. In *Proceedings of the 2nd ACM International Conference on Embedded Systems for Energy-Efficient Built Environments*. ACM, 33–34.
[18] Arka A Bhattacharya, Dezhi Hong, David Culler, Jorge Ortiz, Kamin Whitehouse, and Eugene Wu. 2015. Automated metadata construction to support portable building applications. In *BuildSys*. ACM, 3–12.
[19] Maximilian Christ, Nils Braun, Julius Neuffer, and Andreas W. Kempa-Liehr. 2018. Time Series FeatuRe Extraction on basis of Scalable Hypothesis tests (tsfresh âĂŞ A Python package). *Neurocomputing* (2018).
[20] Laura Daniele, Frank den Hartog, and Jasper Roes. 2015. Created in close interaction with the industry: the smart appliances reference (SAREF) ontology. In *International Workshop Formal Ontologies Meet Industries*. Springer, 100–112.
[21] Bing Dong and Khee Poh Lam. 2014. A real-time model predictive control for building heating and cooling systems based on the occupancy behavior pattern detection and local weather forecasting. In *Building Simulation*, Vol. 7. Springer.
[22] Jonathan Fürst, Kaifei Chen, Randy H Katz, and Philippe Bonnet. 2016. Crowdsourced BMS point matching and metadata maintenance with Babel. In *Pervasive Computing and Communication Workshops (PerCom Workshops)*. IEEE, 1–6.
[23] Jingkun Gao and Mario BergÃĺs. 2018. A large-scale evaluation of automated metadata inference approaches on sensors from air handling units. *Advanced Engineering Informatics* 37 (2018).
[24] Jingkun Gao, Joern Ploennigs, and Mario Berges. 2015. A data-driven meta-data inference framework for building automation systems. In *BuildSys*. ACM, 23–32.
[25] Dezhi Hong, Quanquan Gu, and Kamin Whitehouse. 2017. High-dimensional time series clustering via cross-predictability. In *AISTATS*. 642–651.
[26] Dezhi Hong, Jorge Ortiz, Kamin Whitehouse, and David Culler. 2013. Towards automatic spatial verification of sensor placement in buildings. In *BuildSys*.
[27] Dezhi Hong, Hongning Wang, Jorge Ortiz, and Kamin Whitehouse. 2015. The building adapter: Towards quickly applying building analytics at scale. In *BuildSys*. ACM, 123–132.
[28] Dezhi Hong, Hongning Wang, and Kamin Whitehouse. 2015. Clustering-based active learning on sensor type classification in buildings. In *CIKM*.
[29] Nicolas Hug. 2017. Surprise, a Python library for recommender systems. http://surpriselib.com.
[30] ISO 16739 2014. *Industry Foundation Classes (IFC) for data sharing in the construction and facility management industries*. Standard. buildingSMART.
[31] Srinivas Katipamula, Ronald M Underhill, James K Goddard, Danny J Taasevigen, MA Piette, J Granderson, Rich E Brown, Steven M Lanzisera, and T Kuruganti. 2012. *Small-and medium-sized commercial building monitoring and controls needs: A scoping study*. Technical Report. Pacific Northwest National Lab.(PNNL), Richland, WA (United States).
[32] Merthan Koc, Burcu Akinci, and Mario Bergés. 2014. Comparison of linear correlation and a statistical dependency measure for inferring spatial relation of temperature sensors in buildings. In *BuildSys*. ACM, 152–155.
[33] Jason Koh, Bharathan Balaji, Vahideh Akhlaghi, Yuvraj Agarwal, and Rajesh Gupta. 2016. Quiver: Using Control Perturbations to Increase the Observability of Sensor Data in Smart Buildings. *arXiv preprint arXiv:1601.07260* (2016).
[34] Jason Koh, Bharathan Balaji, Dhiman Sengupta, Julian McAuley, Rajesh Gupta, and Yuvraj Agarwal. 2018. Scrabble: Transferrable Semi-Automated Semantic Metadata Normalization using Intermediate Representation. In *BuildSys*. ACM.
[35] Quoc Le and Tomas Mikolov. 2014. Distributed representations of sentences and documents. In *International Conference on Machine Learning*. 1188–1196.
[36] Xuesong Liu, Burcu Akinci, Mario Berges, and James H Garrett Jr. 2012. An integrated performance analysis framework for HVAC systems using heterogeneous data models and building automation systems. In *BuildSys*. ACM, 145–152.
[37] Yudong Ma, Francesco Borrelli, Brandon Hencey, Brian Coffey, Sorin Bengea, and Philip Haves. 2012. Model predictive control for the operation of building cooling systems. *IEEE Transactions on control systems technology* 20, 3 (2012).
[38] Xiangrui Meng, Joseph Bradley, Burak Yavuz, Evan Sparks, Shivaram Venkataraman, Davies Liu, Jeremy Freeman, DB Tsai, Manish Amde, Sean Owen, et al. 2016. Mllib: Machine learning in apache spark. *JMLR* 17, 1 (2016), 1235–1241.
[39] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. 2011. Scikit-learn: Machine learning in Python. *Journal of machine learning research* 12, Oct (2011), 2825–2830.
[40] Marco Pritoni, Arka A Bhattacharya, David Culler, and Mark Modera. 2015. A method for discovering functional relationships between air handling units and variable-air-volume boxes from sensor data. In *BuildSys*. ACM, 133–136.
[41] Lev Ratinov and Dan Roth. 2009. Design challenges and misconceptions in named entity recognition. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning*. Association for Computational Linguistics, 147–155.
[42] Gerard Salton, Anita Wong, and Chung-Shu Yang. 1975. A vector space model for automatic indexing. *Commun. ACM* 18, 11 (1975), 613–620.
[43] Anika Schumann, Joern Ploennigs, and Bernard Gorman. 2014. Towards automating the deployment of energy saving approaches in buildings. In *BuildSys*.
[44] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *NIPS*. 3104–3112.
[45] Robert Tibshirani. 1996. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)* (1996), 267–288.
[46] Joaquin Vanschoren, Jan N. van Rijn, Bernd Bischl, and Luis Torgo. 2013. OpenML: Networked Science in Machine Learning. *SIGKDD Explorations* 15, 2 (2013).
[47] Ricardo Vilalta and Youssef Drissi. 2002. A perspective view and survey of meta-learning. *Artificial Intelligence Review* 18, 2 (2002), 77–95.
[48] Weimin Wang, Michael R Brambley, Woohyun Kim, Sriram Somasundaram, and Andrew J Stevens. 2018. Automated point mapping for building control systems: Recent advances and future research needs. *Automation in Construction* 85 (2018).
[49] Longqi Yang, Eugene Bagdasaryan, Joshua Gruenstein, Cheng-Kang Hsieh, and Deborah Estrin. 2018. OpenRec: A Modular Framework for Extensible and Adaptable Recommendation Algorithms. In *WSDM*. 664–672.