Full Paper for ACM SenSys Proceedings

Abstract

More than 70% of commercial buildings have giant sensor networks to monitor different aspects of building performance. However, most of these deployed sensor networks have little metadata describing context, which precludes writing analytics applications without familiarity of the particular buildingâĂŹs sensor metadata structure.

In this paper, we propose a technique which learns how to transform a buildingâĂŹs metadata to a common namespace by using a small number of examples from an expert. Once the transformation rules are learnt for one building, it can be applied across buildings with a similar metadata structure. We illustrate this on a testbed consisting of 60 buildings comprising more than 20,000 sensor points. We also illustrate how this common namespace can help a user write analytics applications that do not require building-specific knowledge, and also scales across different buildings.

1 Introduction

Buildings are sites of very large sensor deployments, typically containing up to several thousand sensors reporting physical measurement, continuously. Moreover, with the recent interest in reducing building energy consumption and increasing their efficiency, it is important to consider ways to quickly bootstrap a set of building data streams into an anlytical pipeline. These analyses consist of jobs that help give deeper insight into the overall performance of the building, determine where there are opportunities for energy savings, and discover broken sensors. However, current 'point' naming conventions form a bottleneck in the scalability of the data integration process. A 'point' refers to a physical location where a sensor is taking measurements. Each building vendor uses their own naming scheme and unique variants of each scheme are implemented from building to building; variations exist even across buildings that have contracted the same vendor. This makes the integration process laborious for building experts and a non-starter for non experts. Morever, the process is fundamentally unscalable. If we want to quickly run the job across many sites we need to explore methods quickly normalizing the data for acquisition.

Consider a simple analysis which has the ability to identify anomalous readings from a specific kind of sensor. To execute this job, the process organizes each sensor by type and location, organizes a the distribution of readings across types and locations, and identifies broken sensors where some fraction of their readings are above some threshold value on the distribution. In order to run this application the job needs to know the names of each sensor, its type, and its location. This information is typically encoded in the name of the sensor or 'point', as is common in the building system nomenclature. For example, the point `BLDA1R435__ART` has all the information necessary for analysis in the form of concatinated codes. For example the name of the building (first 4 characters), the air handling unit identifier (the fifth character), the room number (R435), and the type ART (area room temperature) – which indicates that this are measurement produced by a temperature sensor – are all encoded in the aforementioned point name.

However, point names do not always follow the exact same structure and certainly do not follow the same convention across vendors. Because each point is named by a human, the names can vary. This makes it difficult to construct a general set of rules for name construction. However, although name conventions are inexact, they are generally similar across buildings with the *same vendor*, so the rules that describe the name in one building might also work to expand the names in another building. When automatic construction is not possible or uncertain, feedback from the building manager – the expert who can identify the meaning of more cryptic name encodings – could provide feedback about the metadata for the sensor. We make use of automatic name expansion and active leanring techniques in order to learn as much as possible from the set of names avail-

SenSys'13, November 11–15, 2013, Rome, Italy.

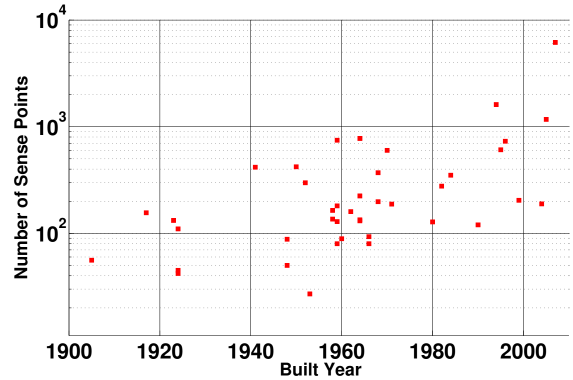
able and combine that with expert feedback to improve the certainty and coverage of our scheme.

In this paper, we propose a set of techniques which learns how to transform a building's metadata to a common namespace by using a small number of examples from an expert. Once the transformation rules are learnt for one building, it can be applied across buildings with a similar metadata structure. We also show how processes that extract statistical features and generate descriptive metadata can help unify data streams with respect to much deeper attributes. We show how our approach makes it easier to write applications across buildings by demonstrating its use by three different applications: 1) a rogue zone detector, 2) a broken senor finder and 3) an application that identifies and ranks the most comfortable rooms. We illustrate these on a testbed consisting of nearly 60 buildings comprising more than 20,000 sense points. We also illustrate how this common namespace can help a user write analytics applications that do not require building-specific knowledge and scales across different buildings. We believe this is an important study given the recent trends in the penetration of the internet of things in our home and our technique can be used to unify that data for broad search and exploration of new applications.

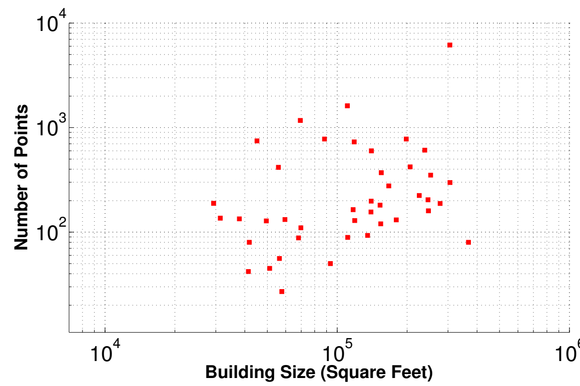
2 Motivation

Buildings are notoriously complex from a management perspective. They consume a large fraction of the energy produced in the United States and much of is wasted [?]. There has been much work in the building science community to reduce their energy consumption and make them more efficieny, but the route to broader impact is typically carried out through regulations guided by the findings of studies in those communities [?]. We aim to let solutions reach buildings *directly* by making sense of the data they produce as quickly and accurately as possible. In order to achieve this at scale, we must explore ways to deal with the data produced from sensors within them and to enable braod anaysis across several buildings at a time. Our study focuses on any building equipped with a network of sensors. Nearly three-quarters of commercial buildings contain a rich sensing fabric, installed as part of the building management system [?]. It is the data from these system and variants of it, that we wish to unify and make sense of in a more systematic and automated fashion.

Fortunately, we have access to a large corpus of data from buildings on our campus. We examine the data from 56 buildings containing over 22,600 sense points. These buildings represent a vast range in age, size, and density of deployment. It also represents deployments that were set up by more than one vendor. As expected, newer buildings have many more sense points than older ones – although some old buildings that have been retrofitted have over 1000 points within them. The general trend in the number of sense points versus the year the building is built is roughly linearly increasing in the log of the number of points, as shown in Figure 1a. The maximum number of points in a single building is 6169 and the minimum is 27. The built years spans over 100 years – from 1905 to 2007. The size range spans over an order of magnitude in square footage from about 30,000 square feet to over 360,000 square feet. There is no observable correlation between the size of the building and the number of sense points. Figure 1b shows a log-log plot of the number of points versus the size of the building. There is one large building with many sense points, but the size seems to have little to do with the density of the deployment. The access this this kind of breadth of system and building types makes our study unique.



(a) Number of Points vs Year Built



(b) Number of Points vs Building Size

Figure 1: Relationship between the built year and the building size on the number of sense points. This data is summarize from a testbed that we used for experiments that consists of almost 60 buildings and over 20,000 sense points.

As mentioned, buildings are notoriously complex and ad-hoc data management practices make it difficult for any analytical solution to be widely ported or run across building systems. For example, consider the following stream names: `BLDA1R435__ART`, `BLDA1R435__ARS`, `BLDA1R545__ART`. Each name encodes contextual information in the form of concatinated character sequences. In these, the first 4 characters refer to the name of the building, the next one encodes the air handling unit association, the next four encode the room, and the last three encode the acronym for the type. Although the examples given are well structured, many variants within the same data set exist. For example, `BLD_1R435_ARG_` is the encoding for a different sensor in the same room as the others, but with a name that is *like* although not exactly the same structure as the others.

When dealing with a small number of points such differences are usually not a problem. Upon visual inspection, the two encodings are similar enough that the engineer can decode the meaning. However, for automatic processing or processing a large number of points, these kinds of variations makes it difficult to generalize the character-contruction rule set. Without rule-set construction the data cannot be ingested properly or interpretted correctly. However, there are "active learning" techniques in the literature [**?**] that address this problem by leveraging the knowledge of an expert. The idea behind active learning is that you can request input from an expert to improve the accuracy of your algorithm. In the case of name/tag expansion you can generalize the set of rules that generate a name/tag by example, iteratively updating the name construction rule set for different types of names. An expert feeds the system examples of names for a particular type of sensor or code, and the algorithms that generalize the character-set construction rules can raise the confidence of the expanded expression. We explore the use of *active learning* techniques to iteratively learn all the variants within an across data sets.

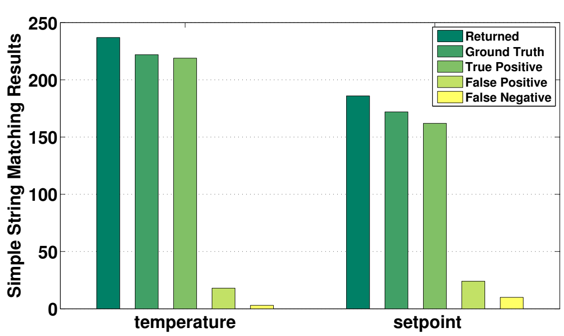


Figure 2: Results when running a grep search on the point names in a single building.

3 Automating Metadata Acquisition

In this section, we go into detail about how we apply program synthesis techniques and the input-output model of interaction to extract enough information from sensor names to enable sufficient coverage of sensor applications. We first provide a general overview of the technique in Section **??**, followed by details of the input required, the synthesis algorithm in Section **??**.

3.1 Inputs, desired outputs, and terminology

The expert is expected to point out *(Tag Name, Tag Value, Value Type)* tuples in the sensor name. A *tag* is mapped on to a substring of the sensor name, which is called its *value*. A tag can have a constant or a variable value. A value should be regarded a constant if it is not specific to that particular sensor.

Sample Input: Suppose the expert is presented with an example `BLDA1R465__ART`, he should qualify it in order as *(site,BLD,const), (ahu,A,const), (ahuRef,1,var), (zone,R,const), (zoneRef,465,var), (zone air temp sensor,ART,const)*. In this example the **site** tag's value is `BLD`, which is not specific to that particular sensor. Hence, the expert should mark it as a constant. On the other hand, the value of the **ZoneRef** tag is specific to that sensor, and hence should be marked as variable.

Output: The synthesis technique should then be able to identify the tags in the sensor name `BLDA5R234___ART`, and output the set of tuples *(site,BLD), (ahu,A), (ahuRef,5), (zone,R), (zoneRef,465), (zone air temp sensor,ART)*. We term each of these tuples as a *qualification*, because it qualifies a set of alphanumeric characters into normalized metadata tags. We term the output as a *full qualification*, if every alphanumeric character in the sensor name was *qualified* by the set of outputed *tags*.

3.2 Synthesis technique overview

String expr **P**	:= **Switch**$((b_1,e_1), \ldots, (b_n,e_n))$
Bool **b**	:= $d_1 \vee d_2 \vee \ldots \vee d_n$
Conjunct $\mathbf{d_i}$	:= $p_1 \wedge p_2 \wedge \ldots \wedge p_n$
Predicate $\mathbf{p_i}$	:= **Match**(v_i,r,k)
Trace expr **e**	:= **SubStr**(v_i, p_1, p_2)
Position **p**	:= **Cpos**(k) | **Pos**(r_1, r_2, c)
Integer expr **c**	:= k (integer const.)
Regular Expression **r**	:= **TokenSeq** $(T_1, \ldots T_n)$
Token **T**	:= C+ | specialToken

Figure 3: Language for learning substring extraction

In this section, we will first describe the high-level logic of the synthesis technique in [**?**] which synthesizes simple regular expressions to transform input columns of a desired spreadsheet to a desired output column. We will then describe the set of specific techniques to adapt this to our context.

The Algorithm:

The main aim of the technique is to learn two sets of information from the inputed examples — (a) whether a string transformation is applicable on a particular input, and (b) what is the set of regular expressions that transform the input string to the output string.

From each user-provided example, the set of all expressions from the language (shown in Figure 3), that could extract the required substring from that input is computed. If there are multiple input-output examples, the substring extraction rules of the multiple examples are intersected to obtain a more concise set of expressions. If the extraction expression sets cannot be intersected, they are maintained as two disjoint sets, which we shall hereby term as a *partitions*.

Finally, for each disjoint set of extraction expressions, a boolean classifier is built in the DNF form, to differentiate the examples in one partition to examples in all other partitions. When a new string is given to this tool, the classifier is applied on it to figure out which partition the new input falls into, and the corresponding set of transformation expressions are then applied on it.

The Language:

The top level expression of the language is the classifier — the **Switch**(b_i, e_i) function, which applies the substring expression e_i to the input only if it matches the boolean expression b_i. The boolean function is in DNF form and is composed of predicates of the form **Match**(v_i, r, k) , which evaluates to true, iff the input v_i has k occurences of the regular expression r.

The Substring expression **SubStr**(v_i, p_1, p_2), evaluates to the substring between positions p_1 and p_2 of the string v_i. **CPos**(k) denotes the position k in the substring. A position expression **pos**(r_1, r_2, c) when applied on a string s evaluates to a position t in the subject string s such that r_1 matches some suffix $s[0..t]$ and r_2 matches some prefix of $s[t...l]$ (where l = Length(s)). Also, t is the cth such match starting from the left end of the string. The regular expressions are either just a single token τ, or a token sequence, **TokenSequence**$(\tau_1..\tau_n)$, or ϵ (which matches the empty string). The tokens τ range over some token classes, e.g one token for alphabetic characters, one for numeric characters, and one for each special character.

3.3 Adapting to our context

The intuition behind adapting this set of techniques to our context is by considering each *tag* as a potential output for a given sensor name. From each given example and for each tag in that example, we can compute the set of all expressions from the language that could extract the required substring from that sensor name. If the same tag is present in multiple examples, the tag's new extraction expression is the intersection of it's extraction expression sets for each of those examples. This may result in disjoint *partitions* for each tag. Finally, a boolean classifier is built to differentiate examples of each partition from all other provided examples.

However, directly adapting this technique to the sensor naming runs into problems. We ran the technique as-is on one of the buildings in our data set, and the results are shown in Figure 4.

We make four design decisions to adapt the algorithm to our setting. First, we augment the set of tokens that regular expressions normally use (one token denoting an alphabetic character, one token denoting a numeric character, and a separate token for each special character), to include one token for each constant value indicated by the expert. We did this because using the non-augmented token set is not expressive enough to extract the desired substrings (as will be shown in Figure **??**). Second,

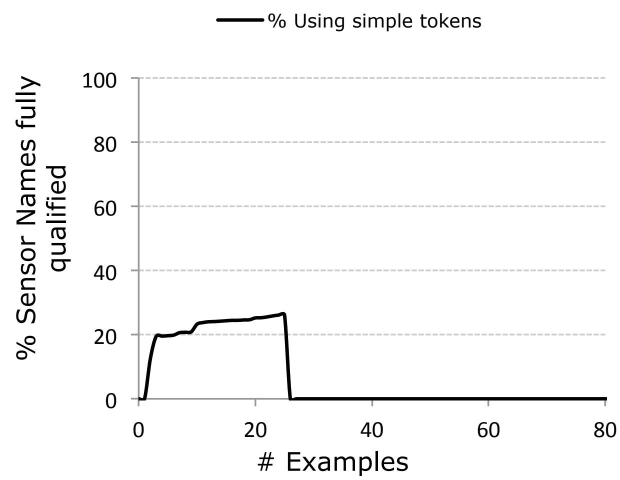


Figure 4: Number of sensor names fully qualified with a token set consisting simply of alphabetic characters [A-Z], numerals [0-9], and one token for each special character.

Second, instead of optimizing for the minimum number of partitions that describe a tag's string extraction rules over all inputed examples, we choose an online approach which just greedily intersects the string extraction expressions from a new example with the existing partition with which it has the maximum overlap. This is done to minimize overhead in an online setting , where the number of existing examples might be large (it takes more than 100 examples to fully qualify all the sensor names on the buildings we tested our technique on).

3.4 Our adaptation

The high-level logic of our algorithm is adapted from [**?**], which implemented learning simple regular expressions to transform input columns to a desired output column in a spreadsheet. The main aim of the technique is to learn two sets of information from the inputed examples — (a) whether a particular tag is applicable on a particular sensor name, and (b) what is the set of regular expression which will give it the alpphanumeric characters corresponding to that tag.

From each expert-provided example and for each tag in that example, we compute the set of all expressions from our language (shown in Figure **??**), that could extract the required substring from that sensor name. The language defines the rules to apply regular expressions to extract required substrings from a string. If the same tag is present in more than one example, the tag's new extraction expression is the intersection of it's the extraction expression sets for each of those examples. If the extraction expression sets cannot be intersected, they are maintained as two disjoint sets, which we shall hereby term as a *partitions*. The intuition behind intersecting the extraction expression sets is that as more examples containing a particular tag is seen, the extraction expressions get more and more concise.

Then, for each disjoint set of extraction expressions for each tag, a boolean classifier is built in the DNF form, to differentiate the examples containing that particular tag, from all the remaining examples. When a new sensor name is seen, the classifier is applied on it to check whether that tag is applicable on it, and if so, then the extraction expressions corresponding to the classifier is applied on the sensor name.

3.5 final output - project Haystack

final output maybe in any form. we realized going into this soon that not all tags conformed to any known schema. For the general tags like room, ahu, vav we have conformed to the project haystack ones. for the remainder of the points which are building specific, we just require that a person uses a consistent schema.

Boosting the metadata can enable better usability across buildings. The question automatically becomes which metadata space to normalize to. There are many metadata schemes devised for representing all buildings. Some of them are too specific and require heavy lifting, and some of them are too simple and do not meet the criteria of being expressive enough for the necessary facets of a building.

The goal is to normalize the existing metadata.

3.6 Technique Overview

The synthesis technique is adapted from [**?**], and tries to learn the regular expressions which

of our technique is to provide building-specific experts the ability to come up with ur technique tried to learn the regular expression patterns that

3.7 learning by example

-The basic structure of our technique is derived from gluwani. -The goal is to

Our proposed technique is derived from the synthesis technique developed in []. In this section, we provide an overview of the technique, and then we will introduce how we adapt this technique to our problem.

Atomic expr $f := \textbf{SubStr}(v_i, p_1, p_2)$
Position $p := k \mid \textbf{pos}(r_1, r_2, c)$
Regular exp $r := \epsilon \mid \tau \mid \textbf{TokenSeq}\ (\tau_1 ... \tau_n)$

As an example, consider that we have to extract `ART` from `BLDA1R465__ART`, the substring expression can be written as $\textbf{SubStr}(s, -3, -14)$, or **SubStr**($s$, **Pos()**

way we learn regular expressions from inputs provided by the expert. We shall use the example scada tag `BLDA1R465 ART` as a goto example throughout this section.

In the following sections, we will use the word *token* to refer a token from a character class. So a token might be is a character or a group of characters in a point to be expanded. In the For instance, in the point `BLDA1R465 ART` has the tokens as shown in Figure **??**

3.7.1 Inputs

For every example an expert provides, we require three types of information - (a) the normalized metadata tag which is contained in the point name, (b) the mapping of the labels in the data to normalized metadata tags , (c) the starting point of those labels, and (d) whether the value of the label is a constant or variable. For instance, consider the expert is asked to fully qualify the sensor point name `BLDA1R435__ART`. Shown below is the expected input by the user.

The aim of the learning algorithm would then be to fully qualify the remaining examples

Whenever the expert types in the explanation for an input, we require that the expert give a full description of the point. The full description of a point consist of the haystack tags the point contains, the starting and ending position of the string that correspond to the haystack tag, mentioning whether the substring is a constant or is variable. For instance, in our example, `BLD` is a constant for the haystack tag **site**, but `465` is a variable substring, because the tag value will change from point to point.

are the tokens contained in a tag, their starting and ending positions, whether the tag has an associated value, and whether the associated value

3.8 challenges

-different types of points all together in the same corpus, and it is not one or two . you have to generate a regex classifier for all known tags. -tokens vary from building to building -> so no pre-defined token (alternative were using Excel's stuff or treating every letter as an individual token). Show the experiment that shows the number of incorrectly qualified vs number of examples added. -simplest classifier to more complex classifier

3.9 technique we chose and modifications

learning by input output example - subtring generation -intersection -predicate generation

4 Evaluation of Learning By Example

In this section, we guage the effectiveness of our learning by example technique by evaluating the number of examples required to qualify labels in two large commercial buildings.

4.1 Test Buildings for Evaluation

We manually generated ground truth data for all the points in two buildings whose building management system was installed by different vendors. Building 1 has 1586 sensor points and was built in the 1990s. Building 2 was built in the 2000s and has 2551 sense points. The label characteristics of the two buildings are shown in Figure 5.

Figure 5a shows that in these two buildings, a few labels (about 20 in each building) frequently appear in a lot of sensor names. This is pretty common in commercial buildings, where a majority of the points are related to zone or room information. For instance, Building 1, has a room setpoint sensor, an airflow sensor and temperature sensor for each of its more than 200 rooms. Each of these points have a label indicating that they are a room, and a room number. These most frequent labels also fully qualify a large number of the sensor names in both buildings. In other words, learning proper classifiers and qualifications for about 20 labels could yield a full qualification for 80% of the sensor names in both these buildings.

The distribution frequency of labels also has a long tail. For both the buildings, the labels corresponding to *site* and *zoneRef* are most common. However, the distribution of labels show that there is also a long tail. These comprise of building specific sensors, alarms or status variables. There are also a lot of incomplete or inconsistent labeling of sensor names, the labels of which fall in this long tail. Thus, one of the main objectives of the learning algorithm is that it does not learn wrong classifiers for labels based on sensor names that fall in this long tail.

4.2 Convergence of labels

4.3 Choosing the Next Example

The large number of sensors in a building pose a challenge in selecting the next example to present to the expert. First, the expert might not always be able to browse through all sensor points to check correct qualification. Also, an expert might visally not be able to discern which points would add the most amount of information to the learning process.

This process can be facilitated by the internal notion the learning algorithm has of how much of each sensor name it has been able to qualify. This notion is maintained by simply comparing the sensor name to all the labels that the algorithm applied on it. Note that, the algorithm can incorrectly apply labels on a sensor name, which may lead to erroneous conclusions. The incorrect application of labels might be due to incorrectness or incompleteness of the boolean matching expression or the string extraction regular expressions.

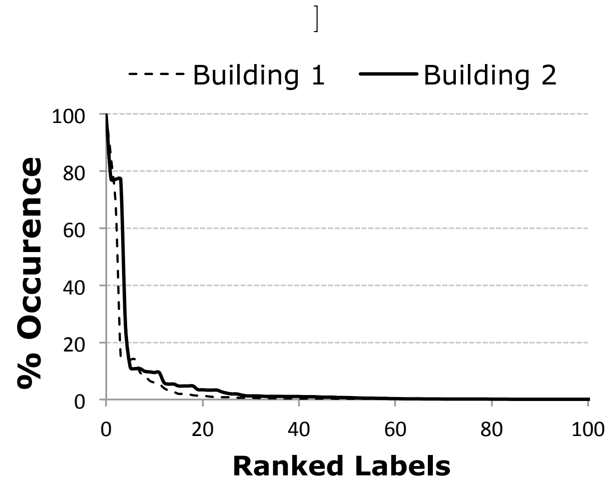
We implemented four different generators to evaluate which example should be provided next to the expert:

Random: This generator just finds at random the next example to present to the expert. While choosing the example, the random algorithm chooses among the set of sensor names which it feels it has not been able to fully qualify.

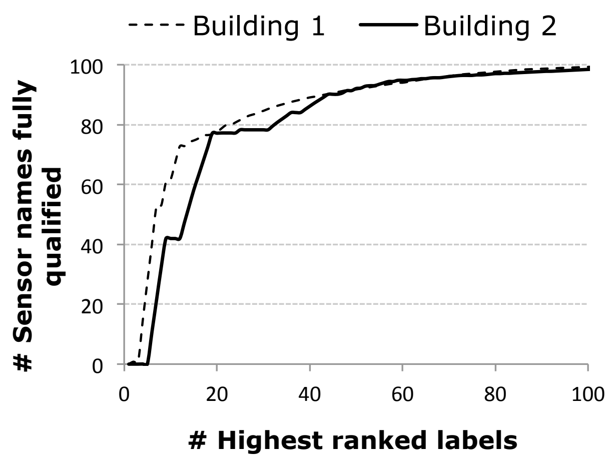
MinRemaining : This generator chooses the example, that according to our tool, has the minimum string length left to qualify. The intuition behind this is to gain more concrete knowledge about a small number of labels.

MaxRemaining : This chooses the example, that the learning technique feels has the maximum string length left to qualify. These examples would help the learning technique gain coverage over the space of unsees labels. The more labels the learning technique knows, the more sensor name information it will be able to qualify.

Self-Correcting : There are some sensor names where the learning algorithm can itself figure out that it has incorrectly qualified a sensor name. There can be three such indicators. First, for a sensor name which has matched its boolean classifier, none of its left or right position regular expressions is applicable. Second, if the sensor name has been qualified with labels that overlap over the same substring. Third, the learning algorithm can get a notion of the labelling uncertainty of a sensor



(a) Percentage of sensor names each label appears in. The x-axis is sorted according to the frequency of occurence of a label



(b) Percentage of sensor names fully qualified by the highest ranking labels. A point (x,y) indicates that y sensor names could be fully qualified by using labels ranked 1 ... x

Figure 5: Characteristics of labels from two buildings we generated ground-truth data to test our learning technique

name, if its qualification labels change drastically when a new example has been added. This generator gives the expert the examples that satisfy the most number of these three criteria. Once, none of the points satisfy these criteria, this generator defaults to the MinRemaining generator.

We wrote a script that automatically gave the learning algorithm the example that it asked for, and evaluated the qualifications provided by the algorithm. We terminated when the number of correct full sensor qualifications reached 70%. Figures 6 show the results of the four generators on the two buildings. The *Random* generator took the least number of examples to achieve full qualification of 70% of the sensor names, achieving it much quicker than the others. The reason for this is due to the long tail of the label distribution as was shown in Figure **??**. The top 20 most occurring labels, by themselves, can fully qualify about 80% of the sensor names. A random generator has a high probability of finding one of these 80% of the points, thus acquanting itself more quickly of the most frequently occuring labels. Neither of the other three classifiers is able to achieve that. They get stuck trying to either get every bit of information from sensor with obscure labels (*MinRemaining*), or trying to cover more labels by first qualifying sensors which have been least qualified(*MaxRemaining*), or by choosing form the set of ill-formed sensor names, which would indicate errors to the learning algorithm (*Self-Correcting*).

4.4 applying to other buildings

5 Related Work

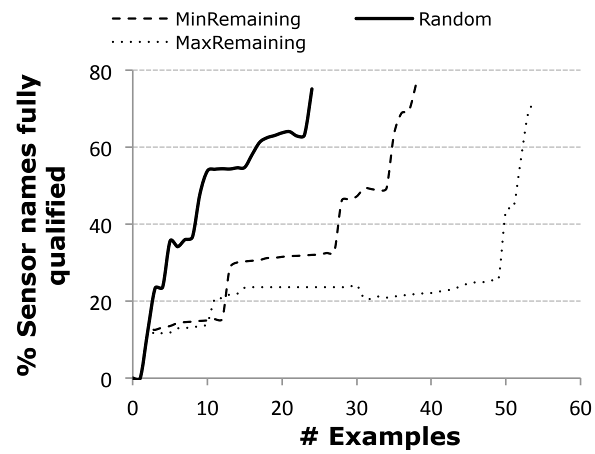
[3] [1] [4] [2]

6 Case Study

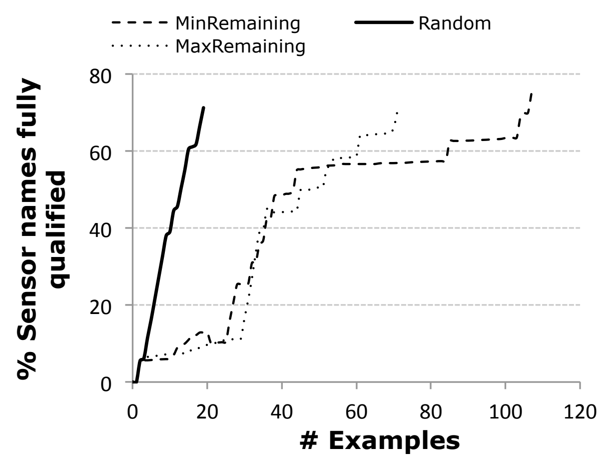
In this section, we demonstrate that with the metadata automatically normalized and expanded using the techniques in previous section, we are able to implement a few applications that are generalizable from one building to another building without modification. As a proof of cncept, we implment two applications: a) identify the uncomfortable rooms and b) detect rogues rooms. We also evaluate the metadata expansion technique in terms of the accuracy for both applications compared against the gorund truth.

6.1 Experimental Setup

We implement two applications and perform the analysis on three buildings from two campuses, and each building is installed with a different management system. Building A and B are from the same campus on the west coast while buidling C is from a second campus on the east coast. Building A was built in mid-1990s using the system from Barrington []. Building B was recently built in 2005 using the Siements BACnet system []. Building C is the newest building among the three installed with Trane's system [], which was put into use from 2011. We collected temperature data as well as setpoint of the rooms in each building. And the data used from each building for analysis is from one week in June 2009, January 2012 and June 2013 respectively.



(a) Building 1



(b) Building 2

Figure 6: The number of examples required to fully qualify 70% of sensor names in two buildings

6.2 Uncomfortable Rooms

It's not unusual to have rooms in a building stay extremely cold or hot thus making the occupants uncomfortable as well as incurring energy waste. The uncomfort are usually caused by improper setpoint settings or dysfunciton of the HVAC systems, and being able to identify these uncomfortable zones or rooms in the building is vital to occupant comfort as well as potential energy savings. With the metadata normalized using our techniques, we are able to search for the desired streams, e.g., the temperature and setpoint of a room, and analyse the thermal performance across different buildings despite of the different naming schema of the sensor or "points".

To identify potential uncomfort in a building, for each room we are particularly interested in a) how much does the temperture deviate from the comfort range? b) how much does the temperature deviate from the setpoint? c) how much does the setpoint deviate from the comfort range? To answer these questions, we first search over the points in each building for distinct temperature stream of each room and the corresponding setpoint. And then we compare the temperature with setpoint as well as the suggested comfort range from ASHRAE [] to compute the temperature deviations in the aforementioned three different perspectives in the one-week period. We accumulate the results from all the rooms per building and generate the CDF as shown in Figure **??**. Each graph illustrates how much the temperature of a building deviates from the comfort range, the setpoint and also, how much the setpoint by itself deviates from the comfort rage. On average, each buidling is uncomfortable to some degree and we rank the rooms in each building by how much they deviates from the comfort zone and the ranking is shown in Table **??**. We also present the ground truth of temperature deviation distribution in Figure 8, where we manually find temperature and setpoint streams for all the rooms to do the same analysis and generate the graphs. The ground truth analysis covers all the rooms in each building so all the potential uncomfortable rooms will be identified while the analysis using the name points expanded with our techniques would miss some of the uncomfortable rooms, since the expansion contains certain error rates.

6.3 Rogue Rooms

Heating and cooling contribute to the largest portion of energy consumption of a buidling, and often, HVAC system operates abnormally either because the system fails itself or the schedule of the buidling is problematic. And there are often some zones and rooms in a building that are constantly cold or hot than their neighbors and incur energy waste. We showed the temeprature deviations above, and we are particularly interested in the periods when a room deviates from the setpoint more than 3 Celsius degree, which is highly likely to indicate that the room is under either heating or coolong. Therefore, for each building, we zoom in to the interested portion and find the rooms whose percentage of time that falls into the interested area is extremely high, i.e., which room's temperature deviates from the setpoint more than 3 Celsius degree in most of the time, and the results are summarized in Table **??**. Again, the grounf truth results are listed in Table **??** as comparison.

BldgA		*BldgB*		*BldgC*	
room#	%	room#	%	room#	%
330B	1	S3-6	1	501	1
340	1	S2-1	1	502	1
420A	1	S1-2	0.93	504	1
420	1	S7-16	0.67	557	1
698	1	S3-13	0.62	511A	1
442	0.996	S4-15	0.62	532	1
398	0.981	S5-5	0.57	224	1
336	0.96	S4-1	0.54	506	1
183	0.92	S5-16	0.48	507	1
498	0.91	S5-7	0.47	511B	1

Table 1: We present the rooms in each building whose temperature deviates from the setpoint more than 3 Celsius degree. For each building, the first column is room number and the second column shows the percentage of the one-week time that the room deviates that much.

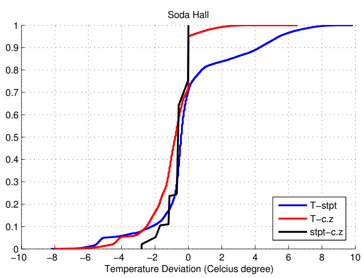
7 References

[1] S. Gulwani. Automating string processing in spreadsheets using input-output examples. In *Proceedings of the 38th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '11, pages 317–330, New York, NY, USA, 2011. ACM.

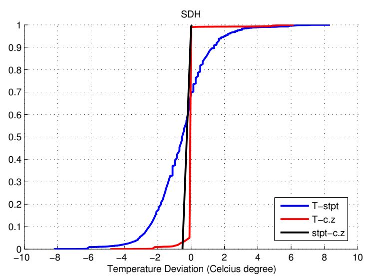
[2] S. Gulwani, W. R. Harris, and R. Singh. Spreadsheet data manipulation using examples. In *In Communications of the ACM*, 2012.

[3] W. R. Harris and S. Gulwani. Spreadsheet table transformations from examples. In *Proceedings of the 32Nd ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '11, pages 317–328, New York, NY, USA, 2011. ACM.

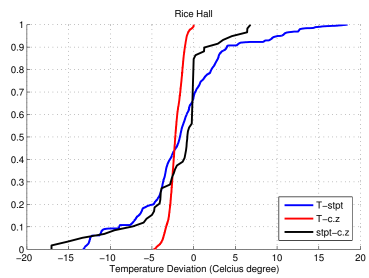
[4] R. Singh and S. Gulwani. Learning semantic string transformations from examples. *Proc. VLDB Endow.*, 5(8):740–751, Apr. 2012.



(a) Building A

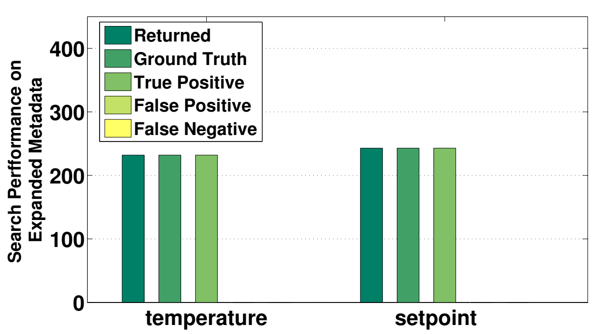


(b) Building B

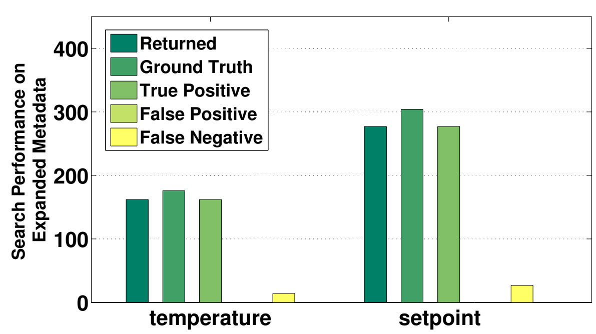


(c) Building C

Figure 7: For each building, we present the distribution of temperature deviation between: a) room temperature and the corresponding setpoint (in blue), b) room temperature and the comfort range suggested by ASHRAE (in red), c) room temperature setpoint and the ASHARE comfort suggestion (in black).



(a) Building A



(b) Building B

Figure 8: The error rates of searches over the expanded metadata using our techniques. We do two searches particularly: "room temp" and "room temp setpoint".