Another Brick in the Wall: Leveraging Feature Extraction and Ensemble Learning for Building Data Classification

Bram Steenwinckel
bram.steenwinckel@ugent.be
Ghent Univirsity - imec
Ghent, Belgium

Abstract

Accurately classifying time series building sensor data is crucial for enabling automation, energy efficiency, and real-time monitoring in smart buildings. The 2024 Brick by Brick competition aimed to develop machine learning models for labeling devices based on building time series data using the Brick schema. Since manually labeling all these devices can require significant effort, the competition sought to automate this process and reduce the workload. In this work, the competition's multi-label classification problem is reframed as a multiclass task in order to apply a classical machine learning approach. Our solution involves a structured pipeline where we extract statistical, frequency-domain, and temporal features from various time series intercals and apply feature selection techniques to enhance the model's efficiency. An extra-trees classifier, optimized using stratified k-fold cross-validation forms the core of our model. Our method achieves a macro F1-score of 0.5767 on the Brick by Brick's public leaderboard test set, demonstrating its effectiveness in automating building sensor classification. These results highlight the value of ensemble learning and structured feature engineering in smart building applications. By reducing manual effort and improving classification accuracy, our approach contributes to more scalable and intelligent building management systems.

ACM Reference Format:
Bram Steenwinckel. 2024. Another Brick in the Wall: Leveraging Feature Extraction and Ensemble Learning for Building Data Classification. In *Proceedings of The Web Conference 2025 (Brick by Brick competition, 2024).* ACM, New York, NY, USA, 7 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Buildings consume about 40% of global energy, making efficient management essential for sustainability [10]. Smart technologies, such as Internet Of Things (IoT) enabled sensors, AI-driven analytics, and automated building management systems (BMS), can optimize energy usage by dynamically adjusting lighting, heating, ventilation, and air conditioning (HVAC) based on real-time occupancy and environmental conditions [9]. Systems leveraging predictive algorithms and machine learning to anticipate energy demand, reduce wastage, and enhance overall efficiency [2]. However, large amounts of data with inconsistent data formats hinder the effectiveness of these approaches [3]. Without standardization, integrating diverse building systems becomes complex, limiting automation, increasing operational costs, and reducing the effectiveness of energy-saving strategies. A standardized approach to classify these IoT building devices is critical for enabling automation, improving interoperability, and ensuring that data-driven decisions can be made efficiently across different platforms and technologies [12].

One promising standardization initiatives is the Brick: a uniform metadata schema for buildings. This metadata standard for building data provides a structured way to represent and categorize sensor and equipment data [1]. However, manually annotating and classifying devices based on building data streams according to this schema is labor-intensive and costly. This limitation has slowed the widespread adoption of smart building technologies, restricting their potential to enhance sustainability and energy efficiency [5].

To address this challenge, the Brick by Brick competition[1] proposes to built an automated solution for classifying the time series building data outputted by these devices using the Brick schema. The competition focuses on developing models capable of categorizing the diverse data streams, such as sensor readings, setpoints, and alarms, into predefined Brick schema categories [7]. This effort aims to streamline building management processes, minimize manual intervention, and unlock new possibilities for data-driven decision-making in smart buildings.

The significance of this task extends beyond the competition itself. The ability to automatically classify building time series data has the potential to change facility management by improving real-time monitoring, fault detection, and predictive maintenance [8]. By fostering the development of advanced classification models, the Brick by Brick challenge contributes to the broader goal of making buildings more intelligent, energy-efficient, and environmentally sustainable.

In this paper, we transformed the competition's original multi-label classification problem into a multiclass classification task to enhance possible model interpretability and performance. To achieve a robust solution, we employed a standardized ensemble machine learning approach, integrating feature extraction, feature selection, and model optimization. The different steps of our pipeline are provided in Section 3, the results are provided within Section 4. We

Brick by Brick competition, 2024,

ACM ISBN 978-x-xxxx-xxxx-x/YYYY/MM
https://doi.org/XXXXXXX.XXXXXXX

[1] https://www.aicrowd.com/challenges/brick-by-brick-2024

provided some additional remarks within Section 5 and conclude this work in Section 6.

2 Competition Background

As stated above, the Brick by Brick competition is a multi-label time series classification challenge designed to automate building devices classification using IoT data.

2.1 Data structure

The dataset comprises time series data collected from three anonymized buildings in Australia, capturing a range of sensor readings, equipment statuses, and environmental parameters such as temperature, setpoints, and alarms. The data is stored as timestamp-value pairs, where timestamps are provided in relative terms. One of the primary challenges is the irregular sampling rates, meaning that data points do not occur at fixed intervals, requiring models to adapt to inconsistencies in time series observations.

To facilitate effective model training and evaluation, the dataset was divided into three distinct subsets:

- Training Set: This portion of the dataset is available for participants to develop and fine-tune their models.
- Leaderboard Testing Set: This subset is used to assess the model's performance during the competition, providing intermediate rankings.
- Secret Competition Testing Set: This final dataset is used to determine competition winners, ensuring unbiased evaluation.

To further improve evaluation robustness, time series chunking is applied, where data is divided into shorter segments ranging from 2 to 8 weeks. This approach allows for assessing model performance across different observation windows, testing their adaptability and consistency.

2.2 Labeling structure

The competition has a hierarchical labeling system based on a modified version of the Brick schema (v1.2.1), encompassing 94-point sub-classes. Label indicators were categorized as follows:

- Positive Labels: The true class of a data point, including its parent categories, providing hierarchical flexibility.
- Zero Labels: Subclasses of the true label that are masked during evaluation, ensuring models do not make unjustified assumptions.
- Negative Labels: Unrelated categories that the model should not predict.

This hierarchical approach encourages precise classifications, allowing for more specific predictions without penalization, thereby fostering nuanced learning.

2.3 Evaluation metrics

The competition is evaluated using a macro F1 score, which ensures balanced performance across all labels, preventing bias toward more frequent classes. The evaluation follows these key steps:

(1) Macro F1 Calculation: Precision and recall are computed for each label while ignoring the masked zero labels.
(2) Label Masking: Predictions more specific than the ground truth are accepted, avoiding penalties for over-specification.
(3) Final Scoring: The final ranking is based on the average macro F1 score across all labels, ensuring fair assessment of different classification strategies.

Additionally, Mean Average Precision (mAP) is used as a secondary metric, considering multiple confidence thresholds to compute precision-recall curves and aggregate scores.

3 Methods

Our solution to the competition follows a classical machine learning pipeline structured into multiple steps, as illustrated in Figure 1. Each of these steps will be examined in greater detail within this section. By the end of the pipeline, we obtained a trained model that, when the feature extraction techniques are applied appropriately, can be used to classify unseen time series.

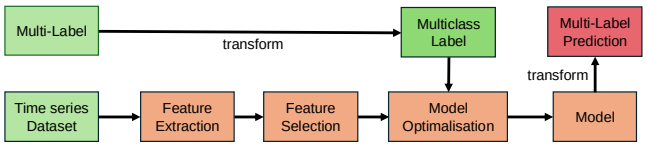

Figure 1: Pipeline existing out of the 3 common machine learning steps of feature extraction, feature selection and model optimization. The original multi-label problem is first transformed to a multiclass setting for enhanced performance.

3.1 Multiclass label

The current labeling structure of the dataset follows a modified version of the Brick schema. Brick is an open-source effort to standardize semantic descriptions of the physical, logical and virtual assets in buildings and the relationships between them. Each sample within our dataset could have multiple labels indicated by the hierarchy structure within the Brick schema. When a sample has the indoor temperature sensor label, it also has the temperature sensor and sensor label accordingly to specify this hierarchical aspect.

This multi-label approach means, theoretically, that each data point can belong to more than one category. Given the fact that this competition had to deal with 94 possible classes, this would result in 2^{94} possible outcomes. However, in practice, when dealing with such a hierarchical structure, the amount of possibilities is much lower. The Brick schema hinders the fact that a sample has both the temperature sensor and, e.g., an alarm label.

Therefore, we identified all the unique label combination based on the training set. We did this by simply concatenating all the label names according to a sample together. By doing this, We get unique labels like: Sensor$Temperature_Sensor$Indoor_Temperature_Sensor. In total, only 91 such unique label combinations are within the training set. Hence the idea to transform the original problem to an 91 multiclass classification problem.

Prediction and evaluation are still be performed on a multi-label aspect. This means that prediction of a multiclass model will have to

be transformed back to individual labels. The algorithm in Listing 1 below is used to transform each multiclass prediction back to its original multi-label format.

Listing 1: Tranforming the multiclass labels back to their multi-label variant.

```
y_pred = pd.DataFrame(np.zeros((#rows, 94)))
for r in range(len(predictions)):
    for value in (predictions[r].split('$')):
        if value != '':
            y_pred.loc[r, value] = 1
```

3.2 Feature Extraction

For the feature extraction in our pipeline, we employ a variety of domain-specific techniques to extract meaningful features from time series data.

3.2.1 Full time series feature extraction. Our pipeline first processes the whole time series to calculate statistical moments like skewness and kurtosis using the scipy stats module [11]. Additionally, frequency domain features, such as spectral entropy and dominant frequency, are derived via the welch method from scipy signal. We further calculate shape-based features like autocorrelation, peak and valley counts, and trend analysis. Next, we defined special handlings for different sensor types: sensor, alarm, setpoint, status, and parameter time series, where unique features like noise level, event density, and outlier ratio were computed based on how these sensor should behave (expert knowledge). Temporal features such as the variance and standard deviation of time differences, inter-arrival times, and periodicity were also extracted. We also included entropy-based features, such as permutation entropy and sample entropy. Additionally, fractal dimension and event-based features (burstiness and inter-event times) were computed to capture the time series' underlying complexity. This multi-faceted approach ensures that the features extracted provide a comprehensive representation of the time series, enhancing the predictive power of the subsequent machine learning models. In total, this resulted in 67 unique features. More details about all the features can be found within the feature extractor files in our repository[2].

3.2.2 Interval-based time series feature extraction. As indicated, all above features were extracted from all available values within a time series sample. As the different time series had a different amount of values and the feature extraction techniques did not take into account the time aspect within these time series, we created a second group of features who more aligned the different time series against each other. For each sample, we used to timestamps to resample the values in averages of 5 minutes, 1 hour, 1 day, 1 week data and calculated the same set of features as in Section 3.2.1 on them. This resulted in $4x67$ additional features.

3.2.3 Timestamp feature extraction. in addition to the primary feature extraction steps, another interesting aspect for analyzing the data involves the consistency and regularity of the time intervals between consecutive values. An alarm time series will provide values at different time intervals compared with a temperature sensor, which can have a fixed sensor rate. To take these differences into account, we calculated the time differences between consecutive timestamps. This is achieved by using the .diff() method on the timestamp column and converting the result to seconds. Next, we assess the most common time difference (mode) across these differences, which is used as the expected interval. By comparing each timestamp's interval with the expected value, we can identify any discrepancies or irregularities in the timing sequence. These discrepancies, or deviations, are captured by flagging time differences that differ from the expected interval. The total number of these deviations, together with the excepted interval is then provided as a feature. We hoped with these two features that sudden gaps in data or other anomalies in the timestamp distribution could be detected.

[2]https://gitlab.com/bram.steenwinckel/brick-by-brick-pipeline

3.3 Feature selection

In total, the previous feature extraction module created 337 features. As they were generated in bulk at once, we were not sure if they were all as relevant for the multiclass problem as expected. Hence, the idea to use feature selection to reduce the amount of features. We utilized the SelectFromModel feature selector from scikit-learn [6]. This method works by fitting a model that uses all the features and then evaluating their importance based on the model's internal feature ranking. Specifically, it calculates feature importance scores and selects the most significant features that contribute the most to the model's predictions. By setting a threshold for feature importance, we were able to retain only the top 40 features, ensuring that we focused on those that had the greatest impact on the model's macro F1 score. We did this to reduce the possible risk of overfitting, enhancing computational efficiency, and improving the interpretability of the results. These top 40 features are available within Appendix A.

3.4 Model optimalisation

We employed the extra-trees classifier as our final model after extensive experimentation with various ensemble methods, including random forests. extra-trees, or extremely randomized trees, is an ensemble learning method that builds multiple decision trees and averages their predictions to enhance accuracy and reduce overfitting. Unlike random forests, which selects optimal split points based on feature values, extra-trees introduces additional randomness by choosing split points entirely at random within a feature's range. We used the scikit-learn implementation in all our evaluations [6].

Listing 2: extra-trees classifier hyperparameters

```
param_dist = {
  'n_estimators': randint(100, 1000),
  'max_depth': randint(10, 100),
  'min_samples_leaf': [2, 4, 8],
  'min_samples_split': [2, 4, 8],
  'criterion': ['entropy', 'gini'],
  'class_weight': [None, 'balanced'],
  'max_features': [None, 10, 50, 100, 250]
}
```

Our choice of extra-trees was guided by internal evaluations, where it consistently outperformed other ensemble methods in terms of predictive performance and stability. We iteratively improved our model using a stratified k-fold cross-validation approach with k=5, ensuring that each fold preserved the distribution of the imbalanced classes. This stratification was crucial in effectively capturing variations in the data and preventing bias toward dominant classes. By tracking performance across multiple iterations, we assessed the impact of additional features, feature selection strategies, and model optimization techniques. We examined the macro F1 score of the multi-labels on the holdout validation sets by transforming back the multiclass labels as stated in Section 3.1. The average macro F1 scores of these holdout validation sets provided a good indicator which configurations led to the most interesting results. This approach also aligned with the leaderboard results on the test set (meaning that an increase in average validation also led to an increased score on the leaderboard). Hence the fact that with a minimum amount of submission, we could already achieve interesting results on an unseen set.

Several parameters of the extra-trees classifier were tuned within this cross validation loop with an additional randomized grid search cross validation of 30 iterations (with 3 cross validation splits). Listing 2 shows the parameters which were tuned. The best parameters for our model were: n_estimators: 314, max_depth: 30, min_samples_leaf: 2, min_samples_split: 4, criterion: entropy, class _weight: None, max_features: None

4 Results

The results on the public leaderboard test set for our approach and configurations are provided in Table 1. The results verifies our incremental strategy. The results were provided first for the full features (which were described in Section 3.2.1). A first interesting solution was provided using a random forest classifier, which was later on switched for the extra-trees classifier as stated above. In the next iterations, we chronologically first integrated the interval features (Section 3.2.2) and time features (Section 3.2.3), and at last reduced the amount of features using the feature selector to select the 40 best performing solutions.

To further analyses our best performing approach, statistics were provided in Figure 2 to present a comprehensive evaluation of the multi-label classification model's performance across different classes. The top-left plot displays per-class metrics, including accuracy, precision, recall, and F1-score. These values vary significantly across different classes, indicating an imbalance in our model's performance. While some classes exhibit high accuracy and precision, others suffer from lower recall, suggesting that the model struggles to identify certain labels effectively. This inconsistency may stem from class imbalance in the dataset or the variation between the test and training set.

The top-right plot presents the average precision (AP) for each class, sorted in ascending order. A clear upward trend can be observed, where a subset of classes achieves high AP values, while others remain low. We suspect that the classes with high AP belong to the main categories within the training set (Sensor, Alarm, Parameter, Status) but no evidence for this claim can be provided. The disparity suggests that while the model performs well for these labels, certain categories are more challenging, potentially due to insufficient training samples or overlapping feature representations with other classes. Some classes are also not detected at all, which might be related to missing label combinations within the training set.

The middle-left plot presents the mean precision-recall (PR) curve across all classes, with a shaded region indicating the standard deviation. This curve offers a broad perspective on the model's balance between precision and recall. As recall increases, the widening shaded region highlights growing variability among classes—while some labels maintain high precision, others experience a notable decline. This suggests the need for further fine-tuning, such as threshold optimization or class-specific adjustments. Additionally, our method's output was constrained to binary indicators. Providing probability scores could offer a more nuanced and comprehensive assessment of performance in this case.

To better understand label interactions, the middle-right plot visualizes the class co-occurrence matrix for the predicted labels only. This heatmap reveals patterns of dependencies between different classes, where certain labels frequently appear together. While some of these co-occurrences may reflect natural relationships within the data, others could indicate systematic model biases. Analyzing these patterns can inform improvements, such as incorporating structured prediction techniques or using multi-task learning to capture label dependencies more effectively. Again, this plot shows there is room for further improvement for several classes.

The bottom row contains box plots of precision and recall distributions across all classes. These visualizations highlight the variation in performance, with a wide interquartile range and the presence of outliers. Again, we see that our model was unable to predict some classes within the test set, giving us evidence that some labels might be underrepresented within the training set.

5 Additional remarks

During our pipeline construction, we pursued several additional optimization strategies aimed at enhancing our model's performance. However, these efforts did not lead to an increased score within our internal validation procedure and, as such, would not have influenced the leaderboard outcomes either.

One of the strategies we explored involved class and sample weighting based on the hierarchical depth of the label. The goal was to give more importance to certain classes depending on their position within the hierarchy, thus potentially improving classification accuracy for more nuanced categories. Despite careful calibration, this approach did not yield any measurable improvement in performance.

We also implemented a multi-stage classification approach. In this method, we first trained a classifier to distinguish between the broader, high-level category classes. Following this initial classification, a more specialized, detailed classifier was employed to differentiate among the finer-grained classes within each broader category. Although this hierarchical breakdown was designed to streamline the classification process and improve accuracy, it ultimately did not translate into better validation scores.

Additionally, we experimented with a hierarchical classification approach using the hiclass Python package, which is specifically

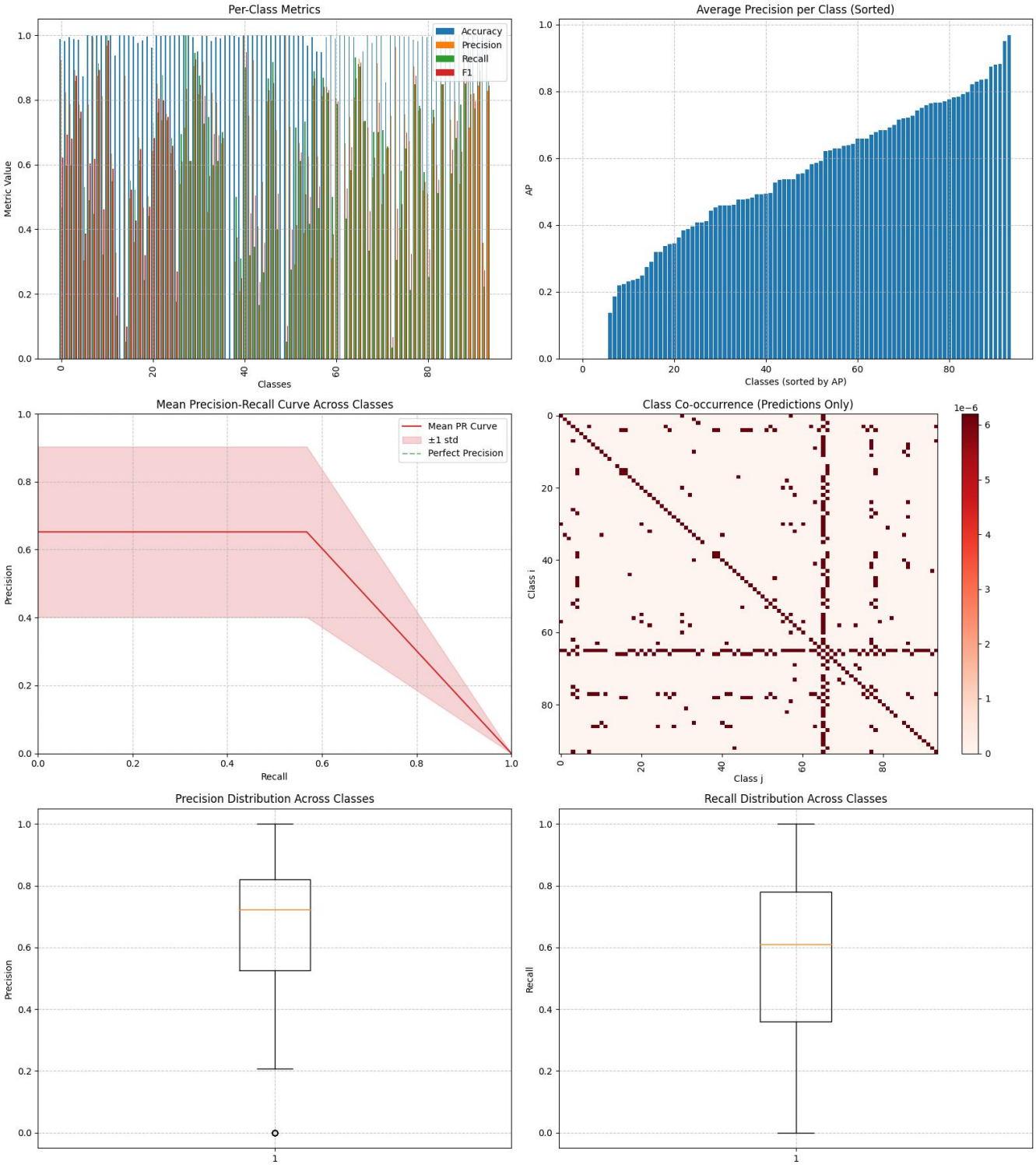

Figure 2: Evaluation of our extra-trees model with top 40 features across the different labels. The top-left plot shows per-class accuracy, precision, recall, and F1-score, highlighting variability in performance. The top-right plot presents the sorted average precision (AP) per class, indicating disparities in predictive confidence. The middle-left plot depicts the mean precision-recall (PR) curve with standard deviation, reflecting performance consistency across classes. The middle-right heatmap illustrates class co-occurrence patterns in predictions, revealing label dependencies. The bottom row contains box plots of precision and recall distributions, emphasizing the spread of model performance across different labels. These visualizations provide insights into the model's strengths, weaknesses, and areas for improvement.

Table 1: Results in Public Leaderboard Set. The main metric is the (macro) F1-score

Method	Accuracy	Precision	Recall	**F1-score**	mAP
Zero	0.9769	0.0000	0.0000	0.0000	0.0000
Random Uniform	0.5005	0.0231	0.4993	0.0356	0.0233
Random Proportional	0.9563	0.0239	0.0224	0.0097	0.0234
Mode	0.9664	0.0000	0.0106	0.0001	0.0000
One	0.0231	0.0231	1.0000	0.0396	0.0231
RandomForest (Features full)	0.9869	0.5108	0.4811	0.4692	0.4061
extra-trees (Features full)	0.9879	0.6200	0.5132	0.5216	0.4884
extra-trees (Features full+intervals)	0.9883	0.6432	0.5567	0.5616	0.5201
extra-trees (Features full+intervals+time)	0.9884	0.6477	0.5587	0.5650	0.5244
extra-trees (Features top 40)	**0.9886**	**0.6521**	**0.5677**	**0.5767**	**0.5310**

designed to handle hierarchical classification problems [4]. By leveraging the structure of the label hierarchy, we aimed to capture relationships between parent and child classes more effectively. However, our results did not improve.

6 Conclusion

In this work, we presented an automated approach for classifying time series building data from the Brick by Brick 2024 competition, transforming the original multi-label classification task into a multiclass problem. By employing a standardized machine learning pipeline that integrates feature extraction, feature selection, and model optimization, we developed a possible robust solution for smart building data classification. Our approach demonstrates the feasibility of leveraging ensemble methods, particularly the extra-trees classifier, to improve classification performance while maintaining computational efficiency. All code to replicate our results was made available at our Gitlab repository: https://gitlab.com/bram.steenwinckel/brick-by-brick-pipeline

Beyond the competition setting, our work contributes to the broader field of smart building management by offering a scalable and standardized data classification method. Accurate automated classification of building time series data enables improved facility management, real-time monitoring, and predictive maintenance, ultimately supporting energy efficiency and sustainability efforts. Future research could explore strategies for handling rare and unseen labels to enhance generalization capabilities. The insights gained from this work provide a solid foundation for advancing intelligent building automation, and we hope our findings encourage further innovation in automated building data classification.

References

[1] 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* 226 (2018), 1273–1292.

[2] Sangkeum Lee, Sarvar Hussain Nengroo, Hojun Jin, Yoonmee Doh, Chungho Lee, Taewook Heo, and Dongsoo Har. 2023. Power management in smart residential building with deep learning model for occupancy detection by usage pattern of electric appliances. In *Proceedings of the 2023 5th International Electronics Communication Conference*. 84–92.

[3] Xiachong Lin, Arian Prabowo, Imran Razzak, Hao Xue, Matthew Amos, Sam Behrens, Stephen White, and Flora D Salim. 2024. A Gap in Time: The Challenge of Processing Heterogeneous IoT Point Data in Buildings. *arXiv preprint arXiv:2405.14267* (2024).

[4] Fábio Miranda, Niklas Köhnecke, and Bernhard Y Renard. 2023. HiClass: A Python Library for Local Hierarchical Classification Compatible with Scikit-Learn." arXiv (Cornell University), December.

[5] Sakshi Mishra, Andrew Glaws, Dylan Cutler, Stephen Frank, Muhammad Azam, Farzam Mohammadi, and Jean-Simon Venne. 2020. Unified architecture for data-driven metadata tagging of building automation systems. *Automation in Construction* 120 (2020), 103411.

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

[7] Arian Prabowo, Xiachong Lin, Imran Razzak, Hao Xue, Emily W Yap, Matthew Amos, and Flora D Salim. 2024. BTS: Building Timeseries Dataset: Empowering Large-Scale Building Analytics. *Thirty-Eighth Annual Conference on Neural Information Processing Systems* (2024).

[8] Mashud Rana, Ashfaqur Rahman, Mahathir Almashor, John McCulloch, and Subbu Sethuvenkatraman. 2023. Automatic Classification of Sensors in Buildings: Learning from Time Series Data. In *Australasian Joint Conference on Artificial Intelligence*. Springer, 367–378.

[9] Aya Sayed, Yassine Himeur, Faycal Bensaali, and Abbes Amira. 2022. Artificial intelligence with iot for energy efficiency in buildings. In *Emerging Real-World Applications of Internet of Things*. CRC Press, 233–252.

[10] Vibhu Sharma. 2024. Integrating renewable energy with building management systems: Pathways to sustainable infrastructure. *Journal of Waste Management & Recycling Technology* 2, 1 (2024).

[11] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. 2020. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods* 17 (2020), 261–272. doi:10.1038/s41592-019-0686-2

[12] Filippo Vittori, Chuan Fu Tan, Anna Laura Pisello, Adrian Chong, and Clayton Miller. 2023. BIM-to-BRICK: Using graph modeling for IoT/BMS and spatial semantic data interoperability within digital data models of buildings. *arXiv preprint arXiv:2307.13197* (2023).

A Forty most important features with their descriptions

The following is a list of the features used in the analysis, with brief descriptions for each:

- **Autocorrelation**: Measures the correlation of a signal with a delayed version of itself, indicating the signal's periodicity and repeating patterns.
- **Sampling Rate**: The frequency at which data is sampled, determining the resolution and accuracy of measurements over time.

- **Event Density**: Represents the frequency of events occurring within a given time period, providing insight into the occurrence of significant moments in the data.
- **Trigger Ratio**: The ratio of triggered events to total events, reflecting the sensitivity of a system to changes.
- **State Length**: Measures the duration of a specific state in a sequence, providing insights into how long the system stays in a particular state.
- **Largest State**: The longest duration state observed, indicating the most prolonged phase in the data.
- **Dominant State**: The state that occurs most frequently or has the largest duration, representing the most common condition.
- **25th Percentile of Time Diffs (seconds)**: The time difference value below which 25% of the time differences fall, offering insights into the distribution of intervals between events.
- **75th Percentile of Time Diffs (seconds)**: The time difference value below which 75% of the time differences fall, indicating the spread of intervals.
- **Median Absolute Deviation (MAD) of Time Diffs (seconds)**: A robust measure of the spread or variability of time differences, useful for detecting outliers.
- **Minimum Time Difference (seconds)**: The smallest observed time difference between events, indicating the most frequent interactions in the data.
- **Permutation Entropy**: A measure of the complexity and irregularity of a time series, reflecting how predictable or random the system is.
- **Sample Entropy**: A measure of the unpredictability of fluctuations in the data, used to analyze time series for complexity.
- **RMS (Root Mean Square)**: A measure of the signal's magnitude or energy, providing an indication of the signal's variability over time.
- **Energy**: Represents the total energy contained in the signal, calculated as the sum of squares of the signal values.
- **Event Density 5T**: Event density measured within a 5-minute (5T) window, providing a localized view of event frequency over short periods.
- **Trigger Ratio 5T**: Trigger ratio calculated over a 5-minute window, showing how often triggers occur within this short time frame.
- **State Length 5T**: Measures the duration of states within a 5-minute window, capturing the system's behavior over short time intervals.
- **Largest State 5T**: The longest state duration observed in the 5-minute window, representing the most stable or enduring state in this short period.
- **Dominant State 5T**: The most common state within a 5-minute window, indicating the predominant system condition in this short period.
- **Permutation Entropy 5T**: Complexity measure applied over a 5-minute window, evaluating how irregular the system is in this localized time frame.
- **RMS 5T**: Root Mean Square applied to data in a 5-minute window, offering a localized measure of signal magnitude in this period.
- **Spectral Entropy 1H**: Measures the complexity of the frequency spectrum of a signal over a 1-hour window, providing insight into signal irregularities in the frequency domain.
- **Event Density 1H**: Event density calculated over a 1-hour window, reflecting the frequency of events over a longer period.
- **Trigger Ratio 1H**: Trigger ratio over an hour-long period, indicating how often events trigger within the 1-hour window.
- **State Length 1H**: Measures how long a state lasts within a 1-hour window, offering insight into system behavior over a longer period.
- **Largest State 1H**: The longest state duration within a 1-hour window, reflecting the most sustained state.
- **Dominant State 1H**: The most frequent state within the 1-hour window, revealing the dominant system condition during that period.
- **Permutation Entropy 1H**: Complexity measure applied over a 1-hour window, evaluating how predictable the system is in this time frame.
- **RMS 1H**: Root Mean Square applied to data in a 1-hour window, capturing the magnitude of the signal over the hour-long period.
- **Smallest State 1D**: The shortest state duration observed over a 1-day period, providing insight into transient conditions.
- **Largest State 1D**: The longest state duration over a 1-day period, highlighting the most persistent state in this time frame.
- **Dominant State 1D**: The most frequently occurring state within a 1-day window, indicating the system's primary condition in a daily context.
- **RMS 1D**: Root Mean Square over a 1-day period, offering a measure of overall signal magnitude over a 24-hour span.
- **Event Density 1W**: Event density over a 1-week window, providing insight into the frequency of events on a weekly scale.
- **Trigger Ratio 1W**: Trigger ratio over a 1-week period, revealing how often triggers occur in a week-long window.
- **Smallest State 1W**: The shortest state duration observed within a 1-week period, capturing fleeting states that might occur infrequently.
- **Largest State 1W**: The longest state duration observed in a 1-week period, identifying the most enduring state over the week.
- **Dominant State 1W**: The most common state within a 1-week window, revealing the predominant condition of the system throughout the week.
- **RMS 1W**: Root Mean Square calculated for data over a 1-week window, providing a view of the signal's magnitude on a weekly scale.