Smart Building Occupancy Estimation through Machine Learning

Maher Dissem, 40262180
Github.com/MaherDissem/Occupancy-Estimation-ML

***Abstract*—Occupancy Estimation (OE) is pivotal in optimizing smart buildings for energy efficiency and resource allocation. This paper explores the integration of IoT sensors and Machine Learning (ML) algorithms to provide real-time insights into occupancy patterns within buildings. Leveraging traditional ML classification models, including logistic regression, decision trees, and naive Bayes, we focus on interpretability while achieving promising performance. To address concerns of redundancy and high dimensionality, Principal Component Analysis (PCA) is incorporated, simplifying the computational process and enhancing interpretability. Our approach is validated through comprehensive model selection, hyperparameter optimization, and benchmarking, resulting in an F1 score of 87% using the Random Forest classifier. This study underscores the efficacy of our methodology in predicting room occupancy, with potential applications in energy optimization tasks such as an advanced HVAC control system based on occupancy levels.**

***Index Terms*—Occupancy Estimation, Principal Component Analysis, Machine Learning, Binary Classification**

I. Introduction

SMART buildings are designed to harness a multitude of IoT sensors to optimize various aspects of building functionality, from energy consumption and environmental control to security and resource allocation. Within this context, Occupancy Estimation (OE) in buildings involves the utilization of IoT sensors in conjunction with Machine Learning (ML) algorithms to provides real-time insights into the presence and movements of individuals in different areas or rooms within a building. It plays an important role in transforming traditional buildings into smart, adaptable, and resource efficient spaces by allowing building managers and facility operators to gain real-time insights into occupancy patterns, helping them optimize building operations such as responsively adjusting heating, cooling, lighting, and ventilation systems based on the environmental conditions and actual occupancy levels, thereby reducing energy wastage. In fact, optimizing the energy consumption of HVAC systems is particularly important for smart buildings, as they account for 50% of a building's energy use [1] and previous research showed that occupant monitoring may account for up to 24% of a building's energy savings [2]. Moreover, the real-time occupancy data gathered from various sensors can be used to optimize the systems used for transporting people, such as escalators and elevators which account for 2-5% of the total energy consumption of large commercial buildings or to adjust water circulation depending on the need and stop water pumps during silent hours [1].

This paper addresses the task of occupancy estimation through ML techniques. [3] reviews several recent ML approaches in this context, leveraging different kinds of sensors. However, in this project we decide to focus on traditional simple ML classification models as they are more relevant to the course curriculum and offer promising performance coupled with interpretability.

In fact, simple ML classification models, such as decision trees or naive Bayes, have been widely studied and proven effective in various domains. Their simplicity often translates into ease of implementation and understanding, making them particularly suitable for applications where interpretability is crucial, such as occupancy estimation.

Moreover, to address the concerns related to potential redundancy and high dimensionality in our feature space, we incorporate Principal Component Analysis (PCA) into our occupancy estimation framework. PCA offers a robust solution to these challenges by effectively capturing the most salient information while reducing the dimensionality of the data. This reduction not only simplifies the computational complexity of the subsequent modeling process but also enhances the interpretability of our results, allowing us to uncover the underlying structures and patterns driving occupancy dynamics more efficiently.

The rest of this paper is structured as follows: In Section II, we present a formal explanation of the PCA algorithm and introduce the machine learning models utilized in our experiments. Section III elaborates on our methodology, detailing the dataset used, data processing steps, and implementation specifics. Additionally, we conduct an Exploratory Data Analysis (EDA) to delve into the dataset, examining variable distributions and correlations. Subsequently, Section IV presents the outcomes of our experiments and the conclusions drawn from them. Finally, Section V discusses these results, addresses the limitations of our approach, and proposes future research directions aimed at further enhancing the efficiency of our methodology.

II. Background

A. Principal Component Analysis

Principal Component Analysis is a widely used dimensionality reduction technique that aims to extract the most informative features from high-dimensional data while preserving the essential structure and variability. It achieves this by transforming the original data into a new coordinate system where the axes are aligned with the directions of maximum variance, called principal components. This transformation is accomplished through the computation of eigenvectors and eigenvalues of the data covariance matrix [4].

The principal components are obtained as linear combinations of the original features. Let X be an $n \cdot p$ data matrix, where n is the number of observations and pp is the number of original features. PCA computes the principal components $Z = [z_1, z_2, \ldots, z_p]$ as: $Z = XV$

where V is the matrix of eigenvectors corresponding to the largest eigenvalues of the covariance matrix of X. Each column of Z represents a principal component, and these components are orthogonal to each other.

The PCA computation process is detailed in Algorithm 1, outlining the sequential steps involved in dimensionality reduction and principal component extraction.

Algorithm 1 Principal Component Analysis (PCA)

Require: Data matrix $\mathbf{X}$, Number of principal components k
Ensure: Principal components $\mathbf{Z}$

1: Standardize the features of $\mathbf{X}$ to have zero mean and unit variance.
2: Compute the covariance matrix $\mathbf{C}$ of the standardized data matrix $\mathbf{X}$: $\mathbf{C} = \frac{1}{n}\mathbf{X}^T\mathbf{X}$.
3: Perform eigenvalue decomposition on $\mathbf{C}$: $\mathbf{CV} = \mathbf{V}\Lambda$, where $\mathbf{V}$ contains the eigenvectors and Λ is a diagonal matrix containing the eigenvalues.
4: Select the k eigenvectors corresponding to the k largest eigenvalues to form the matrix $\mathbf{V}_k$.
5: Project the standardized data matrix $\mathbf{X}$ onto the subspace spanned by the selected principal components: $\mathbf{Z} = \mathbf{X}\mathbf{V}_k$.

B. Machine Learning Classification Models

In this section, we further introduce these models and explain how they operate.

1) Naive Bayes Classifier: The Naive Bayes classifier is a probabilistic model based on Bayes' theorem with the "naive" assumption of feature independence. Given a dataset D consisting of n instances with m features, and a set of k classes $C_1, C_2, ..., C_k$, the goal is to predict the class label y for a new instance x. Bayes' theorem states:

$$P(C_i|x) = \frac{P(x|C_i) \cdot P(C_i)}{P(x)}$$

Where $P(C_i|x)$ is the posterior probability of class C_i given instance x, $P(x|C_i)$ is the likelihood of x given C_i, $P(C_i)$ is the prior probability of class C_i, and $P(x)$ is the evidence probability.

The naive assumption is that the features are conditionally independent given the class, i.e., $P(x|C_i) = \prod_{j=1}^{m} P(x_j|C_i)$, where x_j is the j-th feature of x.

With this assumption, the classifier predicts the class label for x as:

$$\hat{y} = \arg\max_{C_i} P(C_i) \prod_{j=1}^{m} P(x_j|C_i)$$

Where $\hat{y}$ is the predicted class label. Typically, $P(x_j|C_i)$ is estimated using probability density functions, such as Gaussian distributions for continuous features and multinomial distributions for categorical features.

Naive Bayes classifiers are computationally efficient and perform well in many real-world applications, particularly when the independence assumption holds reasonably well.

2) Support Vector Machines: Support Vector Machine (SVM) is a supervised learning algorithm used for classification tasks. Given a training dataset D consisting of n instances represented as feature vectors $x_i \in R^m$ with corresponding binary class labels $y_i \in \{-1, +1\}$, SVM aims to find the optimal hyperplane that separates the data into two classes with the maximum margin. Mathematically, the decision function of the SVM can be defined as:

$$f(x) = \text{sign}\left(\sum_{i=1}^{n} \alpha_i y_i \langle x_i, x\rangle + b\right)$$

where α_i are the learned Lagrange multipliers, b is the bias term, and $\langle x_i, x\rangle$ denotes the dot product between feature vectors x_i and x. The optimization problem for SVM can be formulated as:

$$\min_{\alpha} \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \langle x_i, x_j\rangle - \sum_{i=1}^{n} \alpha_i$$

subject to the constraints:

$$0 \leq \alpha_i \leq C, \quad \sum_{i=1}^{n} \alpha_i y_i = 0$$

where C is the regularization parameter that controls the trade-off between maximizing the margin and minimizing the classification error.

SVM can handle non-linearly separable data by employing kernel functions $K(x_i, x_j)$, allowing it to operate in higher-dimensional feature spaces. Popular choices for kernel functions include the linear, polynomial, radial basis function (RBF), and sigmoid kernels. SVMs are widely used due to their effectiveness in high-dimensional spaces and robustness against overfitting.

3) K-Nearest Neighbor Classifier: K-Nearest Neighbors (KNN) is a simple yet effective non-parametric classification algorithm used in machine learning. Given a labeled training dataset D consisting of n instances represented as feature vectors $x_i \in R^m$ with corresponding class labels y_i, and a new instance x to be classified, KNN predicts the class label of x by a majority vote among its k nearest neighbors in the feature space. The class label of x is determined by:

$$\hat{y} = \text{argmax}_{y_j} \sum_{i=1}^{k} w_i \cdot I(y_i = y_j)$$

where $\hat{y}$ is the predicted class label for x, w_i is a weight assigned to the i-th nearest neighbor (often equal weights are used), and $I(\cdot)$ is the indicator function that returns 1 if the condition is true and 0 otherwise. The choice of k is a crucial parameter in KNN, as it affects the model's bias-variance trade-off. A smaller value of k leads to more flexible decision boundaries but may increase the variance due to noise, while

a larger k reduces variance but may introduce bias. Distance metrics such as Euclidean distance or Manhattan distance are commonly used to measure the similarity between instances in the feature space.

KNN is intuitive, easy to implement, and suitable for datasets with complex decision boundaries, but its computational complexity grows linearly with the size of the training dataset, making it less efficient for large datasets.

4) Decision Tree: Decision Tree is a widely-used machine learning algorithm for classification tasks, characterized by its hierarchical structure of decision nodes and leaf nodes. At each decision node, the algorithm selects a feature and a threshold to partition the data into subsets based on the feature's value. This process is recursively applied to each subset until certain stopping criteria are met, such as reaching a maximum tree depth, minimum number of samples per leaf, or when further splits fail to improve the model's performance.

The decision tree can be represented by the following equation:

$$h(\mathbf{x}) = \sum_{i=1}^{N} w_i \cdot \mathrm{I}(\mathbf{x} \in R_i)$$

where $h(\mathbf{x})$ is the predicted class label for the input vector $\mathbf{x}$, N is the number of leaf nodes, w_i is the class label associated with leaf node i, and R_i is the region defined by the decision boundaries leading to the leaf node i. The function $\mathrm{I}(\mathbf{x} \in R_i)$ is an indicator function that returns 1 if $\mathbf{x}$ belongs to region R_i and 0 otherwise.

The algorithm for constructing a decision tree for classification is outlined below:

Algorithm 2 Decision Tree for Classification using Gini Index

Require: Training dataset $\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$, Maximum tree depth D

Ensure: Decision tree classifier $h(\mathbf{x})$

1: Initialize the root node.
2: Recursively partition the data at each node by selecting the best feature and threshold according to a splitting criterion, such as Gini impurity: Gini($\mathbf{X}$) = $1 - \sum_{k=1}^{K} p_k^2$, where p_k is the proportion of samples belonging to class k in dataset $\mathbf{X}$.
3: Stop partitioning if the tree depth exceeds D or if the node contains fewer samples than a specified minimum.
4: Assign the most frequent class label in the leaf node as the predicted label.

This algorithm iteratively constructs the decision tree by selecting the best split at each node based on a chosen criterion until a stopping condition is met. The resulting tree can then be used to predict the class labels of unseen instances by traversing the tree from the root node down to a leaf node based on the feature values of the instance.

5) Random Forest: Random Forest is an ensemble learning method widely used for classification tasks in machine learning. It operates by constructing a multitude of decision trees during training and outputting the mode of the classes (classification) or the mean prediction (regression) of the individual trees.

Let D denote the training dataset consisting of n instances with m features and corresponding class labels y_i. Random Forest builds multiple decision trees by employing a technique called bootstrapped aggregation or bagging. Each tree is trained on a random subset of the training data and a random subset of features, which helps to decorrelate the trees and reduce overfitting.

Given a new instance x to be classified, the Random Forest predicts its class label based on the majority vote of all the individual trees. Mathematically, the prediction $\hat{y}$ of Random Forest can be represented as:

$$\hat{y} = \mathrm{mode}(y_1, y_2, ..., y_{\text{num\_trees}})$$

where y_i represents the prediction of the i-th decision tree in the forest. In addition to reducing overfitting, Random Forest also provides estimates of feature importance based on how much the accuracy of the model decreases when a particular feature is randomly permuted.

This information can be valuable for feature selection and understanding the underlying data relationships. Random Forest is known for its robustness, scalability, and ability to handle high-dimensional datasets with mixed data types. Moreover, it requires minimal hyperparameter tuning, making it a popular choice for various real-world applications.

III. Methodology

A. Dataset

We have selected the dataset from [5] as our primary data source. Originally curated for occupancy estimation in smart buildings, this dataset consists of time series data recorded within an office space (office H358) at the Grenoble Institute of Technology that accommodates four individuals. The data was collected over nearly three years, spanning from 2015 to 2018 and using various non-intrusive sensors including humidity, temperature, power consumption and co2 concentration meters and with varying sampling rates, ranging from seconds to a few minutes.

B. Data processing

We initially acquired raw SQL dataset comprising time series records from more than 30 IoT smart sensors deployed throughout the building under study. Initially, we filtered the sensors to focus solely on those pertinent to the occupancy data of the H358 professor's office. These sensors generated observations at varying time intervals, resulting in an irregular sampling rate. To standardize the dataset, we resampled it at 30-minute intervals by averaging multiple observations within each timeframe. Subsequently, we organized the observations from different sensors along the column axis, aligning them with the new timestamps. We omitted sensors with a high percentage of missing values ($\geq 40\%$) and discarded data rows containing any NaN values, indicating sensor inactivity during those periods.

TABLE I: Correlation of the room occupancy status with other sensors.

Sensor	Correlation
CONTACT_8	0.83
CONTACT_9	0.53
POWER_10	0.50
POWER_17	0.22
ILLUMINANCE_15	0.22
TEMPERATURE_16	0.19
ILLUMINANCE_23	0.17
TEMPERATURE_18	0.13
POWER_5	0.05
POWER_6	0.02
POWER_12	0.01
POWER_22	0.00

C. Exploratory Data Analysis

The data processing process yielded a refined dataset comprising 22,498 records for 13 sensor features including power, temperature, illuminance and contact sensors. We select Occupancy Estimation as the target variable whose values we want to predict from the records of the other 12 sensors.

This variable is binary, i.e. the sensors used for creating this dataset only indicates whether the room is occupied or not, not accounting for the number of person in the room. Figure 1 illustrates the distribution of this dataset, revealing a significant imbalance issue where the room remains empty approximately 75% of the time. This skew may pose a challenge for machine learning classification models, especially those biased towards the majority class.

Additionally, Figure 2 depicts the fluctuation of the target variable and sensor readings over a random typical day. From this observation and data exploration for other days (not display in this report for the sake of conciseness), we find out that the room tends to be occupied on weekdays, particularly in the afternoon. Interestingly, we observe a lot of variations in sensor values, but they don't seem intuitively connected to occupancy status. To delve deeper and provide more rigorous insight into this, we present a correlation heatmap encompassing all available data in Figure 3 where correlation is computed as follows:

$$r = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum_{i=1}^{n}(X_i - \bar{X})^2}\sqrt{\sum_{i=1}^{n}(Y_i - \bar{Y})^2}}$$

We find out that sensors of the same type (temperature, contact and illuminance) seem to may be correlated while the target variable is mostly correlated with the contact and power sensors. Correlation of the Occupancy status with other sensors is displayed in Table I.

Furthermore, Fig. 4 illustrate box plots for all the descriptive variables. It can be observed from the figure that most of the sensors records are skewed and that the dataset contain a lot of outliers. However, these data points are reported by the dataset authors to be part of the normal sensor behavior under regular condition, hence, we do not filter them out.

D. Experimental Setup

Following initial data processing, we directly input the data into the machine learning models without additional pre-processing. The dataset is partitioned into training and testing sets, with a 20% ratio.

We define for each model a set of possible hyperparameters to tweak. We employ a grid search approach to optimize these hyperparameters, exploring all possible combinations within a cross-validation framework on the training data. Subsequently, we evaluate the performance of the best model configuration on the test data.

Our experimentation involves integrating this pipeline with Principal Component Analysis (PCA) for various numbers of components. Through analysis of the explained variance rate, we determine that the optimal performance for PCA is achieved with 3 components, as elaborated in Section IV-A.

From a technical standpoint, we only employ standard Python libraries: We leverage scikit-learn, which offers a high-level API for training, inference, and evaluation of machine

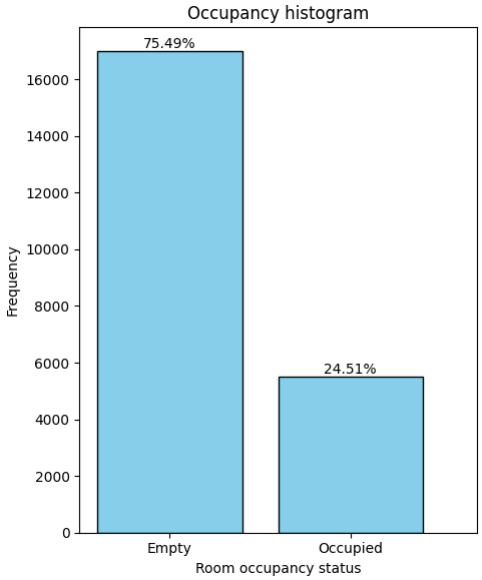

Fig. 1: Distribution of the target variable.

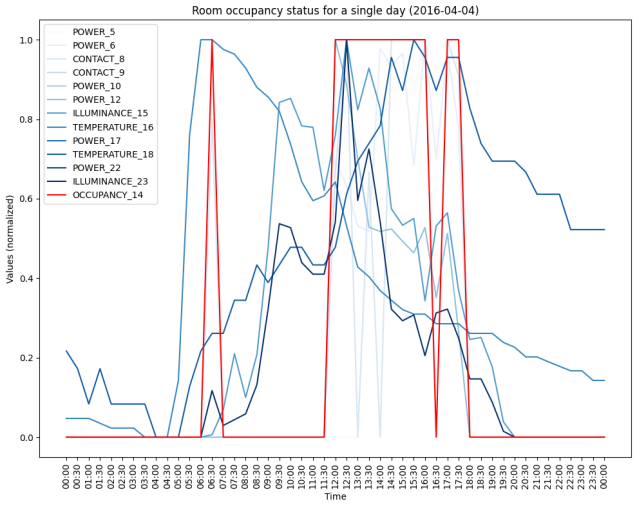

Fig. 2: Evolution of each of the sensors' data throughout a random day. Values are min-max normalized.

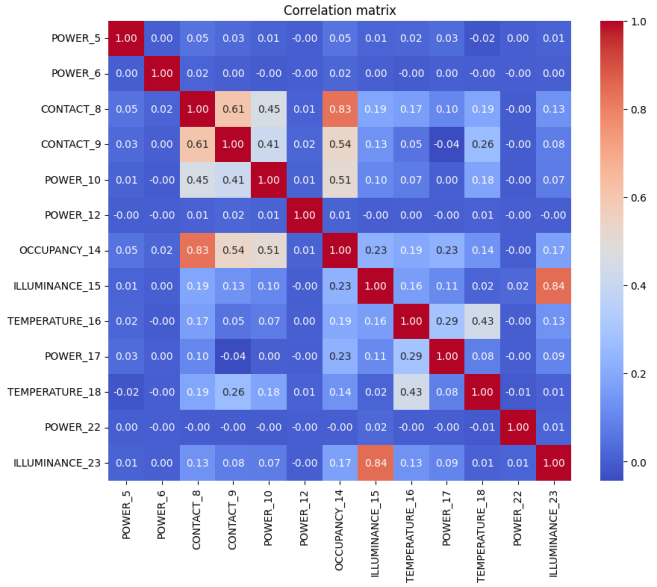

Fig. 3: Feature correlation heatmap.

learning models and for conducting PCA, Numpy and Pandas for data manipulation, Matplotlib for visualization and MLflow To log and track experiment results.

E. Evaluation Metrics

In our experiments, to evaluate and compare the performance of the various classification models, we leverage the following metrics:

- **Accuracy (ACC)**: Accuracy measures the proportion of correctly classified instances (timestamps where the occupancy status is predicted right) among the totasl instances. It is calculated as:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

  where TP denotes true positives, TN denotes true negatives, FP denotes false positives, and FN denotes false negatives. In imbalanced data domains such as our case, where one class significantly outweighs the other, accuracy alone may not provide a reliable measure of model performance because a model could achieve high accuracy by simply predicting the majority class for all instances, while performing poorly on the minority class. Therefore, to gain deeper insights into the model's effectiveness, we introduce additional metrics such as precision, recall, F1-score, and AUROC, which offer a more nuanced evaluation by considering the true positives, false positives, true negatives, and false negatives across both classes.
- **Precision (P)**: Precision measures the proportion of correctly predicted positive instances among the total instances predicted as positive. It is calculated as:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

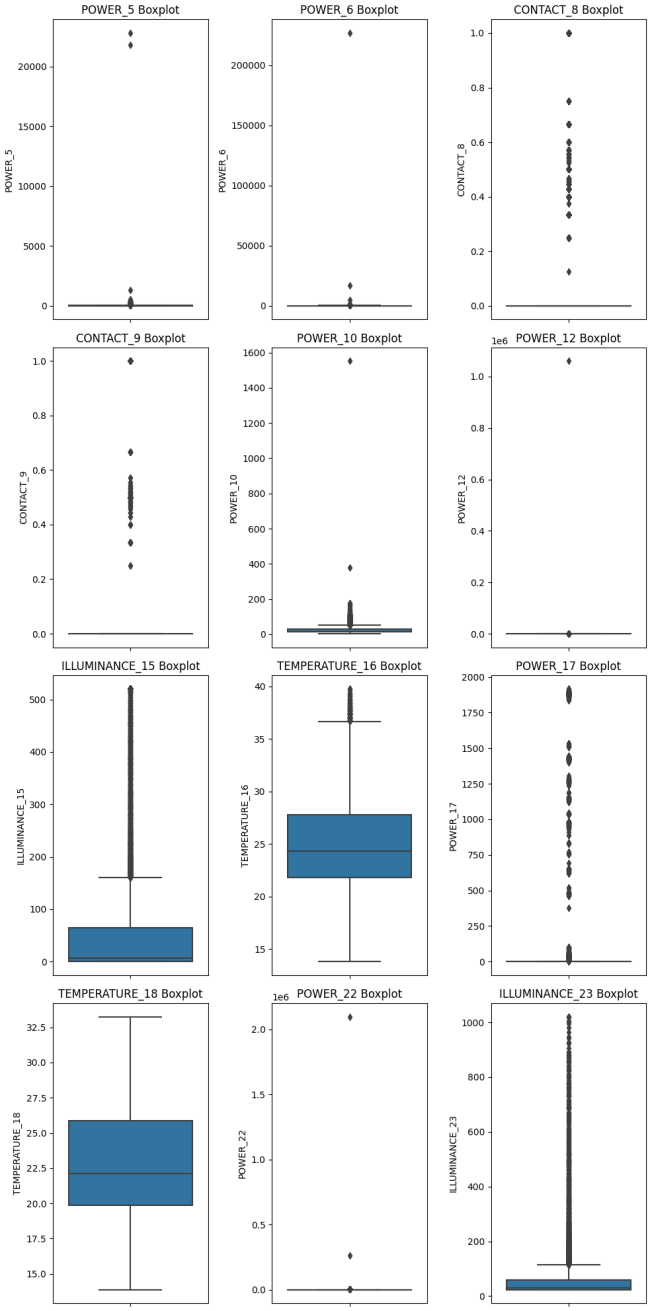

Fig. 4: Box plots of the descriptive variables.

A high precision means that when the model predicts a space as occupied, it is likely to be correct, minimizing false alarms.

- **Recall (R)**: Recall, also known as sensitivity or true positive rate, measures the proportion of correctly predicted positive instances among all actual positive instances. It is calculated as:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

A high recall means the model is good at identifying occupied spaces, minimizing instances where occupied spaces are flagged as empty.

- **F1-score**: The F1-score is the harmonic mean of precision and recall, providing a balanced measure between them. It is calculated as:

$$\text{F1-score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

- **Area Under the Receiver Operating Characteristic Curve (AUROC)**: This metric assesses the model's ability to discriminate between occupied and empty spaces, regardless of the threshold used for classification. It measures the probability that the score of a positive example is higher to negative sample's. It ranges from 0 to 1, where a higher AUROC value indicates better performance while an AUROC of 0.5 suggests a model with no discrimination ability (equivalent to random guessing).

IV. Experiments and Results

A. PCA Results

The original $n \cdot p$ dataset is reduced through PCA. Each column of the eigenvector matrix A is represented by a PC that captures an amount of data that determines the dimension. The obtained eigenvalues are as follows:

$$\begin{bmatrix} 7.91 \times 10^{-1} \\ 1.99 \times 10^{-1} \\ 9.19 \times 10^{-3} \\ 1.82 \times 10^{-4} \\ 1.64 \times 10^{-4} \end{bmatrix}$$

We apply PCA to reduce the number of features from 12 to only 3, corresponding to the first 3 eigen values that are considerably larger than the rest.

While this may seem drastic, we visualize in Fig. 5, the percentage of variance explained by different number of components and find that employing 3 principal components keeps 98% of the total variation of the data.

Fig. 6 represents the PC coefficient plot. It visually represents the amount of contribution each feature has on the first two PCs. The figure supports the previous calculation of PCs, and it can be clearly seen from the figure that the POWER_5 and POWER_12 sensors contribute the most to the first two PCs, while the variance of the other sensors seems to be explained by the third PC.

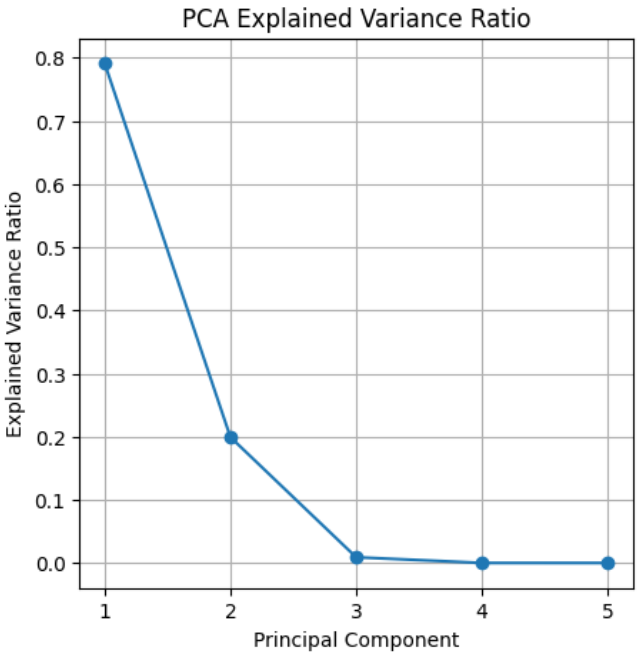

Fig. 5: PCA scree plot.

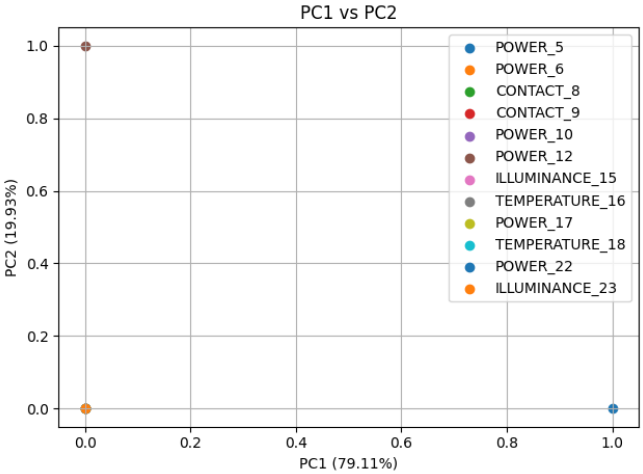

Fig. 6: PCA coefficient plot.

B. Classification Results

Table II presents the results obtained by various Machine Learning models post PCA. Across the board, we observe high accuracy among all models. However, given the metric's sensitivity to the inherent data distribution of the target variable (where a naive model solely predicting empty rooms achieves 75% accuracy), we prioritize other evaluation metrics.

Specifically, the Naive Bayes model exhibits low recall, indicating a high rate of false negatives (i.e., failure to detect occupied rooms). The Support Vector Machine classifier falls into only predicting the majority class, this can be concluded from its 75% accuracy, coupled with a 0% $F1$ score. Conversely, the remaining models perform comparably well, with a slight edge observed in the Random Forest model. This model achieves an $F1$ score of 87% and an AUROC of 90%. We also display its ROC curve in Fig. 8 and confusion matrix in Fig. 7 which indicate that the model performs very well in deed with very few false positive and false negative classifications.

Moreover, to provide further analysis of the performance impact of employing PCA, we compare the performance observed of the Random Forest model with and without run-

TABLE II: OE Benchmark of various ML models

Model	Accuracy	F1	Precision	Recall	AUROC
Naive Bayes	0.85	0.58	0.93	0.42	0.70
SVM	0.75	0.00	0.00	0.00	0.50
KNN	0.93	0.85	0.92	0.78	0.88
Decision Tree	0.93	0.85	0.90	0.80	0.89
Random Forest	0.94	0.87	0.92	0.82	0.90

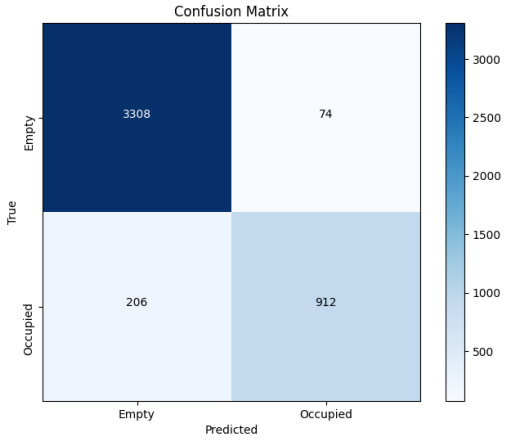

Fig. 7: Confusion matrix of the Random Forest model trained on PCA transformed data after hyperparameter optimization.

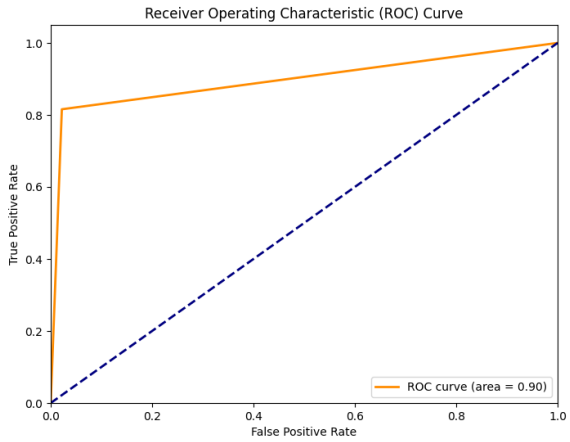

Fig. 8: ROC curve for the Random Forest model trained on PCA transformed data after hyperparameter optimization.

TABLE III: Comparison of a Random forest model trained on PCA transformed data versus on raw data

Model	Accuracy	F1	Precision	Recall	AUROC
RF + PCA	0.94	0.87	0.92	0.82	0.90
RF on raw data	0.99	0.97	0.97	0.97	0.98

ning PCA (both after a separate hyperparameter optimization process) in Table. III. We clearly observe that incorporating PCA significantly reduces the performance of the model. Nonetheless, PCA offers advantages in occupancy estimation in smart buildings, such as dimensionality reduction, which simplifies computation and storage requirements, especially beneficial for resource-constrained environments like edge computing. Additionally, PCA aids in noise reduction and feature selection, enhancing model interpretability and generalization capabilities.

V. Discussion

Through a rigorous comparison of the different ML models, we conclude that the Random Forest model yields the optimal occupancy estimation performance.

PCA indeed offers significant advantages in terms of data storage and computational efficiency, particularly crucial for edge computing scenarios. However, when more powerful hardware resources are available, it becomes pertinent to consider utilizing all available features and implementing more elaborate feature engineering techniques. This could involve leveraging human expertise to select optimal sensors or employing automated processes such as neural networks for data selection. Moreover, given the nature of this data, there's potential to further enhance performance by exploiting temporal dependencies within the features. Hence, we intend to explore modern time series feature engineering methodologies like temporal convolution, recurrent neural networks, or attention mechanisms.

While our current approach with traditional machine learning models yields reasonable performance, adopting more complex models holds the promise of achieving even better results. Therefore, in a future research, we aim to experiment with modern machine learning models and neural network architectures to further enhance performance.

VI. Conclusion

In this project, we employ the machine learning classifier to predict the occupancy status of a room given a set of non-intrusive sensors covering features such as power consumption, temperature, illuminance and contact. The model selection is made after training, hyperparameter optimization, and benchmarking a set of different ML models. Furthermore, leveraging PCA as part of our pipeline ensures that we can effectively manage model complexity with a minimal performance cut while extracting meaningful insights from our occupancy estimation task. With only three components, covering 98% of the data variance, we achieve an $F1$ score of 87% using the Random Forest classifier, highlighting the effectiveness of our approach and potential for widespread adoption in the building sector for energy optimization tasks such the control of HVAC systems according to a room's occupancy.

References

[1] ABB, "Energy efficiency of smart buildings: Towards zero consumption and beyond," *White paper: ABB Motion*, 2021, retrieved May 1, 2024. [Online]. Available: https://www.energyefficiencymovement.com/wp-content/uploads/2021/05/ABB_EE_WhitePaper_Smart-buildings_final-1.pdf

[2] S. H. Kersken M, "Simulation study on the energy saving potential of a heating control system featuring presence detection and weather forecasting," *Internal report of Fraunhofer Institute*, 2013.

[3] Z. Chen, C. Jiang, and L. Xie, "Building occupancy estimation and detection: A review," *Energy and Buildings*, vol. 169, pp. 260–270, 2018.

[4] H. Abdi and L. J. Williams, "Principal component analysis," *Wiley interdisciplinary reviews: computational statistics*, vol. 2, no. 4, pp. 433–459, 2010.

[5] M. Amayri, A. Arora, S. Ploix, S. Bandhyopadyay, Q.-D. Ngo, and V. R. Badarla, "Estimating occupancy in heterogeneous sensor environment," *Energy and Buildings*, vol. 129, pp. 46–58, 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0378778816306223