ML Tool for Assisted Design of Timber Buildings

Malak Lahlou Nabil, Sara Anejjar, Antoine Schutz
Department of Data Science, EPFL, Switzerland

***Abstract*—This report introduces a Machine Learning tool aimed at automating and enhancing the structural design process for timber buildings, particularly light-frame structures. It utilizes ML to predict structural specifications and safety assessments based on basic architectural and location data from a dataset of 200 buildings [1] . The classification part achieved an 90%-98.6% accuracy using the Decision Tree Classifier, while the regression component employed an ensemble of Random Forest, Bagging, and Gradient Boosting Regressors, yielding results between 0.0009 and 0.98 Mean Absolute Error for various prediction tasks.**

I. Introduction

In the structural design sector, especially regarding light-frame timber buildings, there is a major challenge: traditional design methods are heavily reliant on manual labor and significant resources. This is particularly problematic for buildings in earthquake-sensitive areas, where the balance between safety and efficiency is critical. A solution would be to use Machine Learning (ML), promising to revolutionize the building design process. By applying ML algorithms, we can automate structural design, making the process not only safer and more efficient but also more cost-effective.

The goal of this project is to create an ML-powered tool designed for predicting structural details and conducting safety evaluations in timber buildings. With 200 light-frame timber structures, this tool aims to leverage existing building information to generate initial structural design estimates. The project objectives are : 1. To use basic building information to estimate detailed structural specifications (Type C). We separated this task into two models: one that uses information about each wall, and the other will predict T(x) and T(y) and uses information about the building. 2. To create an ML model that uses information of Type A, B and C to estimate safety evaluations (Type D). 3. To streamline the design process for timber buildings, reducing the time and resources typically required for manual design while maintaining high safety standards.

II. Data Description

The provided data is organized across eight Excel files, each representing a subset of buildings categorized by architectural configurations and connection types.

- Type A: Basic Architectural and Location Information : Contains building details like the number of stories, location, construction materials, and foundation soil type.
- Type B: Architectural Design Details : Focuses on architectural planning outcomes, including room layouts and structural wall placements.
- Type C: Specifications for Structural Walls: Includes wall specifics, including dimensions and materials used.
- Type D: Safety and Performance Assessment : Uses Types A, B, and C data to evaluate building safety and inform decision-making for engineers.

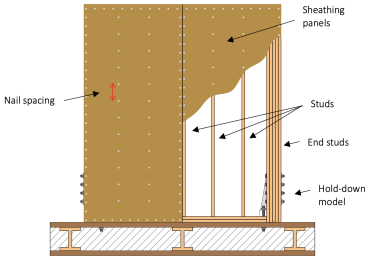

Fig. 1: Physical meaning of information Type C for structural walls.

For our first model, we have a total of 62263 rows, which means that there are 62263 walls, sum of walls in each direction, each story and each building. For our second and third model, we have a total of 200 rows, which corresponds to the total number of buildings. We already know that all features that we can find in the header of the buildings are categorical. We wanted to know if there are more categorical values especially for those we will predict. We found that we can have three other categorical values : Nail Spacing [cm], Number Sheating pannels and the Number end studs.

III. Data Pre-Processing

The goal of our pre-processing steps is to convert raw data from various Excel files into a clean, structured format that can be easily used for building machine learning models. By handling missing values, extracting and organizing relevant information, and ensuring data consistency, we're setting a foundation for any subsequent task.

1. **Parsing header information:** We divide the header into segments and extract information such as the building's architectural style, the number of stories, soil type, seismic zone, and connection method. This breakdown is essential for understanding and categorizing the data.

2. **Filling Missing Values:** When working with Excel files, when cells are combined, only the first cell holds the actual value, while the rest have empty values ("NaN"). In this dataset, the observations follow a logical order, especially for the number of stories and direction. So, we filled in missing values by using the value from the previous observation to keep the dataset organized. We also filled some missing values

manually directly on the raw files, but only for the direction and story which also follow a logical order.

3. **Data Matching:** We have diverse information distributed across multiple files. For instance, we require performance data (Type D information) for each building from a separate dataset. To achieve this, we cross-reference and extract performance data using the header information from the primary dataset. But it was mostly data matching inside each file depending on the building, story, direction and wall. Since the files were discontinuous, which contained multiple tables beginning at distinct indices within the same file, we used precise indexing.

4. **Adding a feature:** To evaluate safety and performance of a building, we believe that the total number of walls may help predicting better, since this information will be lost after the feature selection. Indeed, not all buildings have the same number of walls, however they also don't have the same number of stories. (It actually did not help, probably because there are too many features or the information is irrelevant).

IV. Feature Engineering

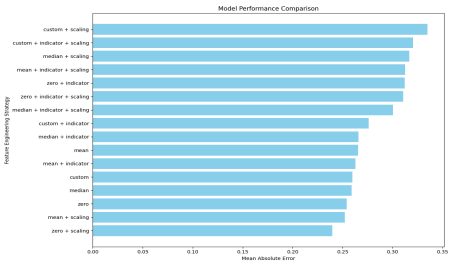

Fig. 2: Comparison of MAE for different methods to deal with missing Nan values

1. **Dealing with missing values:** When exploring the data, initially there were no real data missing values, which means they were created by us when we did the pre-proceesing. Indeed, here each Nan value represents the absence of a wall, so the best result was replacing the nan values with zero, median is not far off.

- Impute with Zero: It seems logical to replace Nan with 0, indicating no wall or a wall of zero size. This is appropriate if our model interprets a zero value as the absence of a wall.
- Impute with Median/Mean: Replace missing values with the median of each feature can be useful if we assume that the impact of adding another wall to the building is justified by the fact that we add another story to this building. These new walls don't necessarily improve any performance since there are other stories.
- Custom imputation : if there were two many missing values, we replaced them with zero else with the median.
- Creating Indicator Variables for Columns with Missing Values : Helps keep track of which data points had missing values in specific columns and thus help predict better performance values.

2. **Dealing with categorical values:** One-hot encoding converts categorical variables, it creates binary columns for each category which is essential for models that can only interpret numerical values.

3. **Scalling:** Scaling ensures that all features contribute equally to the result. It's especially important for algorithms that calculate distances between data points and for optimization algorithms used in neural networks.

4. **Feature selection:** We observe that a combination of these methods yields the best results

- Dropping Columns with Zero Variance: Such features do not contribute any information that can be learned by a model, and removing them can reduce model complexity and computational cost.
- Dropping Columns Based on a Threshold for Most Common Value: Threshold: 95 percent. Columns where a high percentage of the rows have the same value might not be very informative. Dropping these columns can prevent the model from being biased towards this frequent value and can improve generalization.
- Remove highly correlated columns from the DataFrame : Highly correlated features often contain similar or redundant information. Keeping both features doesn't provide additional information, .

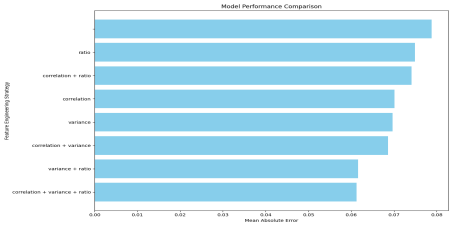

Fig. 3: Comparison of MAE for different feature selection methods

V. Models and Methods

A. Implementation of Machine Learning Models

For this project, we need to predict multiple values about the specifications for the walls (type C) and multiple values about the building's safety (type D). For this purpose, we tried multiple models to understand which one is adapted to our problem, specifically neural networks and classical ML methods. We decided on using MSE as a loss function and MAE as a metric for regression tasks, given their ability for precision and robustness. Furthermore, using the same metrics across different models allow us to compare our implementations easily. Since the values for categorical variables were balanced, we used Accuracy.

1. **Multi-output neural networks**

Neural networks are known for automatically learning hierarchical representations of data. Especially when dealing with diverse features related to wall specifications and building safety, as neural networks can capture patterns and dependencies that traditional models might find it challenging to discern. One of

the main reasons as to why we chose to try neural networks is also its ability to apply different machine learning methods to each output, which was one of our requirements since some values are categorical and others are continuous.

2. **Classical Ml Methods**

Classical machine learning algorithms play an important role in structured data analysis, avoiding the complexities of deep learning networks. Techniques such as **Decision Trees**, **Linear Regression**, and **Gradient Boosting Regressors**, alongside their classification equivalents, are effective in managing well-defined features and clear data relationships. In our project, these classical ML techniques offered initial insights before diving into Neural Networks, with the added benefits of simpler implementation and enhanced interpretability.

Our notebooks were structured in two parts: The first part concentrated on modeling predictions for continuous variables using **LinearRegression**, **Ridge**, and **Lasso** for their linear modeling and regularization capacities. We also integrated **BaggingRegressor** and **GradientBoostingRegressor** for their adeptness in handling complex, non-linear data patterns.

The subsequent section addressed categorical variables, employing **LogisticRegression**, **RandomForestClassifier**, **DecisionTreeClassifier**, and **KNeighborsClassifier**, selected for their effectiveness in multi-class outcomes and complex decision boundaries.

B. Testing and Tuning

1. **Neural Networks:**

The training of neural networks involves various hyperparameters, and we explored multiple configurations to optimize model performance using cross-validation through GridSearch. To enhance training efficiency, we incorporated callbacks, such as `ReduceLROnPlateau`. This dynamic adjustment of learning rates during training helps navigate the model toward convergence, improving the overall training process. All these implementations need hyperparameter tuning which are composed of the following:

- `learning_rate`: Determines the step size during optimization.
- `activation`: Introduces non-linearity to the model.
- `epochs`: Defines the number of training iterations.
- `batch_size`: Specifies the number of samples in one iteration.
- `factor`: Reduces learning rate when performance plateaus.
- `patience`: Epochs with no improvement before LR reduction.
- `min_lr`: Lower bound on learning rate during training.

In neural networks, finding the best number of layers alongside with their size is the most important task. We achieved this by implementing a Keras tuner that systematically optimizes and explores the hyperparameters of our neural network.

- `num_layers`: Determines the number of layers in the neural network.
- `units_<i>`: Sets the number of units in the i-th dense layer.
- `activation_<i>`: Introduces non-linearity to the i-th dense layer.
- `l1` and `l2`: Applies L1 and L2 regularization to the dense layer's weights.
- `leaky_alpha_<i>`: Defines the slope for Leaky ReLU in the i-th layer.
- `dropout_<i>`: Randomly omits a fraction of units in the i-th layer for regularization.

2. **Classical ML Methods:**

We focused on hyperparameter tuning to optimize model performance. For ensemble methods like RandomForestRegressor, BaggingRegressor, and GradientBoostingRegressor, we adjusted `n_estimators` (10, 20) to find a balance between computational efficiency and prediction accuracy. A higher `n_estimators` value typically enhances model performance but at the cost of increased computational load. For KNeighborsRegressor and KNeighborsClassifier, we tuned `n_neighbors` (1-9) to mitigate over-sensitivity to training data noise or excessive smoothing. In Ridge and Lasso regression, `alpha` (0.1 - 10.5) controlled regularization strength, balancing model bias and variance. This hyperparameter optimization, through Grid Search across a broad parameter spectrum, identified the most effective settings for each model, ensuring robust and accurate predictions.

VI. Results

We evaluated the performance of the outputs across three key domains: Wall Specification (Type C), Building Parameters (Type C), and Building Safety (Type D).

- **Wall Specification (Type C)** For the first outputs specific to walls, as there were categorical values, we added outputs with regression model and others with categorical one.

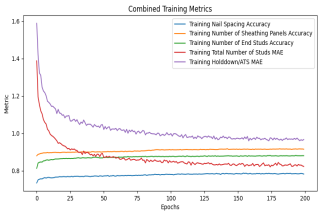

Fig. 4: Loss training plot for Wall Specification (Type C)

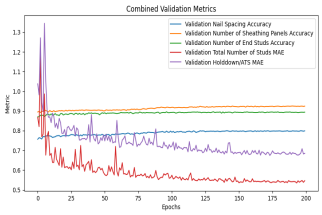

Fig. 5: Loss validation plot for Wall Specification (Type C)

Parameter	Baseline	Classical Methods	Neural Network
Nail spacing (cm)	22%	90%	78.3%
Nb sheathing panels	60%	97.6%	91.6%
Nb end studs	12%	98.6%	88.1%
Total nb studs	0.9291	0.14	0.82
HoldDown / ATS	2.6134	0.009	0.96

For neural networks, the optimal number of layers is three (128, 64, 128). Dropout is applied in the first layer with a rate of 0.1, indicating a slight regularization to enhance model generalization and L1 and L2 regularization are both applied with small strengths (1e-05), preventing overfitting

by penalizing large weights. The regularization parameters are particularly useful when handling categorical and continuous values because they prevent overfitting and balance the model complexity.

For classical models, for the first outputs specific to walls, as there were categorical values, we added outputs with regression model and others with categorical one. The Best Model for categorical Variables was DecisionTreeClassifier.

Building Specification (Type C) This model has two continuous outputs specific to buildings.

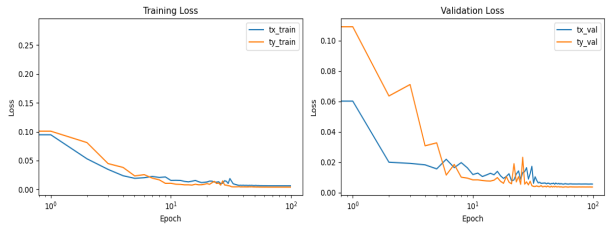

Fig. 6: Loss training plot for Building Parameters (Type C)

Parameter	Baseline	Classical Methods	Neural Network
Tx(s)	0.0528	0.047	0.054
Ty(s)	0.0432	0.038	0.045

For neural networks, the optimal number of layers is three (64, 96, 64). ReLu and swish activation functions have been used to promote non-linearity and smoothness to the model. This latter incorporates regularization as well. This complete configuration gives advantage to robustness for these regression tasks. The results obtained by Classical ML methods are significantly better than the results obtained via Neural Networks.

Building Safety (Type D) This model has many outputs and takes as inputs types A, B and C.

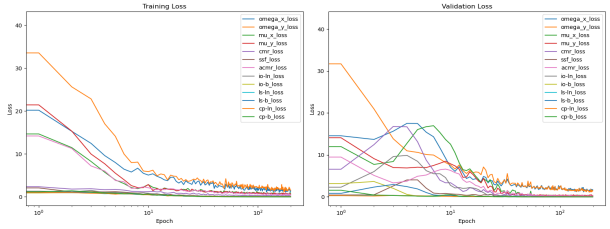

Fig. 7: Loss training plot for Building Safety (Type D)

For neural networks, the optimal number of layers is three (64, 64, 96). Using ReLU and swish activation functions as well for robustness and non-linearity, the model also incorporates regularization techniques such as L1, L2 and Dropout to prevent overfitting.

For classical models, in Ridge regression, the alpha parameter controls the strength of L2 regularization. The best alpha value was for the lowest alpha we tried : 0.1, which means less regularization, this allows the model to fit the data more closely. This shows that the data did not suffer significantly from overfitting, and thus, a minimal level of regularization was sufficient. This balance suggests that our dataset likely had a good structure and complexity where the benefits of regularization were achieved without the need for a high alpha value.

Parameter	Baseline	Classical Methods	Neural Network
Omega x	3.130	0.984	0.954
Omega y	3.949	0.948	0.953
Mu x	0.586	0.227	0.439
Mu y	0.474	0.234	0.354
CMR	0.363	0.244	0.271
SSF	0.019	0.009	0.069
ACMR	0.572	0.323	0.338
IO-ln	0.056	0.050	0.121
IO-b	0.038	0.014	0.040
LS-ln	0.058	0.048	0.107
LS-b	0.041	0.019	0.042
CP-ln	0.055	0.049	0.102
CP-b	0.048	0.022	0.044

VII. Discussion

In the investigation of neural network methodologies applied to a dataset consisting of 200 buildings, we had difficulties in achieving optimal model performance. An important consideration is the limitation in dataset diversity, as the information from the 200 buildings might not be sufficient to cover the entire range of patterns and variations that could exist in different structures. As they are usually known, neural networks often demonstrate their full potential on large datasets where they can learn complex patterns and generalize well. For classsical machine learning methods, we believe that these models yielded the best results due to several reasons. RandomForestRegressor was effective for regression as it is handles non-linear relationships and interactions between features. BaggingRegressor reduces variance and overfitting since it uses the concept of ensemble learning, where multiple models are trained on different subsets of the data. GradientBoostingRegressor worked fine since it builds trees one at a time, where each new tree helps to correct errors made by previously trained trees. For classification, DecisionTreeClassifier might have performed best thanks to its ability to make no assumptions about the distribution of data, which can be advantageous if the actual data distribution is unknown or complex. Decision Trees can also model interactions between the features well, which is beneficial in classification tasks.

VIII. Summary

Navigating the complexities of structural design in timber buildings, our study applied classical machine learning techniques with a focus on models like Decision Trees and Gradient Boosting Regressors. These methods, known for their efficacy in structured data, delivered solid initial results and better interpretability compared to neural networks. Our project was structured in two stages: one for continuous variable predictions and another for categorical data analysis. Our comprehensive approach in model selection and hyperparameter tuning played an important role in achieving great predictions and it reaffirms the strength of classical ML approaches in practical and complex applications.

IX. ETHICS

Overview: In the pursuit of enhancing engineering practices through Machine Learning (ML) models tailored for assisting in building design, our project acknowledges and seeks to mitigate potential ethical risks associated with the utilization of such technologies. Our assessment, conducted using the Digital Ethics Canvas [2], underscores several critical considerations across various domains.

Non-Maleficence: While our solution doesn't foresee direct harmful applications, a potential risk identified lies in the consequences of erroneous outputs. Inaccurate or flawed computations could lead to severe implications such as building collapse due to low-quality design or unnecessary costs resulting from overcompensation in design alterations.

Our project prioritizes robustness and accuracy to prevent these negative impacts. However, fully eliminating this risk remains challenging due to the complex nature of building design.

Privacy: The solution primarily deals with building information rather than personal data. However, there's a nuanced concern regarding the revelation of internal building structures, potentially infringing on privacy. Despite our commitment to data security and limited access to the information solely within our team, the risk of inadvertent disclosure cannot be entirely dismissed.

Dataset: The dataset used in our project, compiled by Xavier Estrella, was intended to explore the feasibility of ML-assisted engineering design. However, the lack of clarity regarding data collection methods, contributors involved, and storage location poses a risk of unaccounted biases or incomplete understanding of the dataset's origins and implications.

Conclusion: Using the Digital Ethics Canvas, we saw that the main risks of this project lies in the Non-Malefience part, errors from the solution can result in either structural failure if the generated outcomes are excessively lenient or unnecessary expenses if the outcomes are overly strict.

REFERENCES

[1] https://www.sciencedirect.com/science/article/abs/pii/S0141029621013134
[2] digital-ethics-canvas