Why do buildings collapse? A predictive and descriptive analysis of earthquake damage

Álvaro Jesús Bernal Caunedo
Dpto. Lenguajes y Sistemas Informáticos
Universidad de Sevilla
Sevilla, España
alvbercau@alum.us.es

Álvaro González Frías
Dpto. Lenguajes y Sistemas Informáticos
Universidad de Sevilla
Sevilla, España
alvgonfri@alum.us.es

Antonio Rodríguez Ruiz
Dpto. Lenguajes y Sistemas Informáticos
Universidad de Sevilla
Sevilla, España
antrodrui2@alum.us.es

Adrián Romero Flores
Dpto. Lenguajes y Sistemas Informáticos
Universidad de Sevilla
Sevilla, España
adrromflo@alum.us.es

***Abstract*—This project analyzes the 2015 Gorkha earthquake's impact in Nepal using data from 4,000 buildings of various construction types. Key features include age, height, and structural attributes. Through exploratory data analysis, dimensionality reduction techniques like PCA and t-SNE, and machine learning methods, we identified factors influencing building damage. However, challenges such as low interpretability, feature variability, and unbalanced data hindered distinguishing damage grades 2 and 3. While clustering methods revealed patterns, insufficient data granularity limited predictive accuracy. This study highlights the need for enriched datasets to improve earthquake damage prediction and recovery strategies through data-driven approaches.**

***Keywords*—Earthquake, Prediction, Machine Learning, Buildings**

I. Introduction

The dataset selected in this project is derived from the extensive damage assessment conducted following the 2015 Gorkha earthquake in Nepal. This earthquake, which struck on April 25, 2015, with a magnitude of 7.8, resulted in widespread destruction across the region, affecting over 8 million people and leading to significant economic losses estimated at around $7 billion. The earthquake caused severe damage to approximately 500,000 buildings, highlighting the urgent need for effective post-disaster recovery strategies.

The data collected encompasses detailed information on 4000 buildings, including various construction types such as reinforced concrete, brick masonry, and stone masonry. This dataset provides valuable insights into the impacts of the earthquake on different building structures and their vulnerabilities.

Key features included in the dataset consist of building-specific attributes such as age, height, land surface condition, house position, plan configuration, foundation type, ground floor type, roof type and superstructure type.

This analysis was developed as part of a Data Mining project for a Machine Learning Olympiad (2024)[1]. The comprehensive nature of this dataset allows for an in-depth examination of the factors influencing building damage and recovery needs. By leveraging advanced machine learning techniques, the study aims to enhance the accuracy of damage predictions and inform effective rehabilitation interventions. This effort not only contributes to improved urban resilience and disaster recovery planning but also serves as a benchmark for integrating data-driven solutions into post-disaster recovery strategies globally.

II. Methodology

A. Exploratory Data Analysis

This part is explained in detail in `01. Exploratory Analysis.Rmd`

The exploratory data analysis (EDA) phase was crucial for understanding the dataset, identifying patterns, and gaining insights into the predictive task. This phase involved examining the dataset's structure, distribution, and relationships between variables to inform subsequent preprocessing and modeling steps. The EDA process was divided into several key components:

1) Data Overview: After loading the dataset, we conducted a preliminary examination to understand its structure and contents. The dataset consisted of 36 columns and 4000 rows, with each row representing a building's characteristics and damage grade. The target variable, *damage_grade* , was categorical and represented the severity of damage caused by an earthquake with an integer value in the range *[1,3]*. The dataset also contained several categorical and numerical features, each providing potentially valuable information for the predictive task.

[1]https://www.kaggle.com/competitions/ml-olympiad-predicting-earthquake-damage/overview

2) Data Consistency: Next step was to check the consistency of the data. We tried first to identify missing values and potential inconsistencies that could affect the quality of the predictive model. The dataset was found to be relatively clean, with no missing values in the columns as shown in Figure 1.

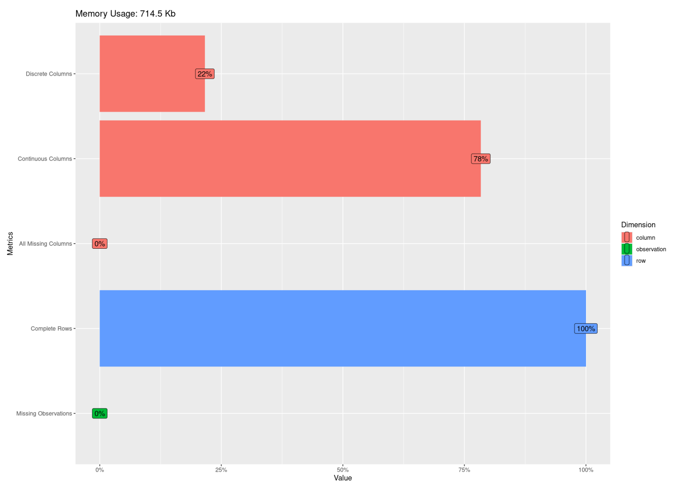

Fig. 1. Data distribution

B. Data visualization

To gain deeper insights into the data, we decided to visualize the distribution of values for the different attributes available, as some of then, in particular the categorical were hard to interpret as their values were mostly given by single character. Such was the case of columns *position* or *surface_condition*.

We started by analyzing two features which represented percentages: *height_percentage* and *area_percentage*. These two appeared to be already normalized, as their values were in the range *[0,1]*. The distribution of these two features was visualized using boxplots, which showed that the majority of buildings had a height percentage of around 0.2 and an area of around 0.1. In terms of height there weren't many outliers, but in terms of area there were many buildings with values higher to that of the majority. However we decided to let those values as they were because it would make sense that in a city there would be buildings with a larger area than others. We did however infer that since the height of the buildings was mostly the same, that would mean that most buildings were either houses or small buildings.

Next we analyzed the other numerical features, which were *age*, *count_floors_pre_eq*, *count_families*. And as we suspected, most buildings were residential houses, as the majority, with a familiy or two at most (Figure 5), rarely more that that. We also found that most buildings were registered as being 0 years since construction, and there was also a spike at 30 years since construction, which could mean that wither there was a

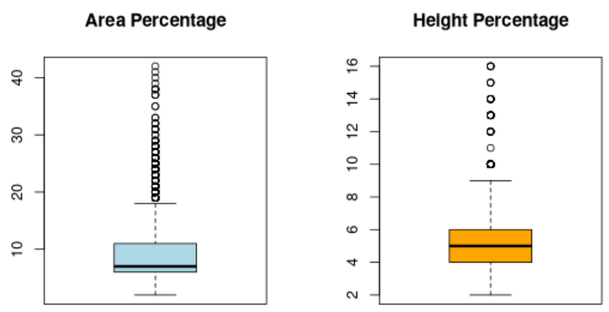

Fig. 2. Height and Area percentage data distribution

boom in construction 30 years ago, or that the data was not properly recorded.

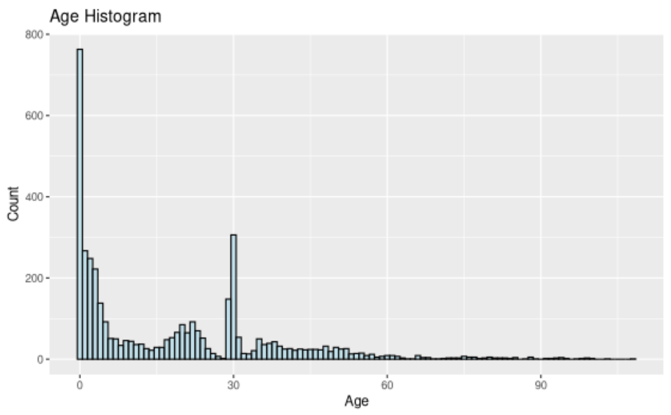

Fig. 3. Building age histogram

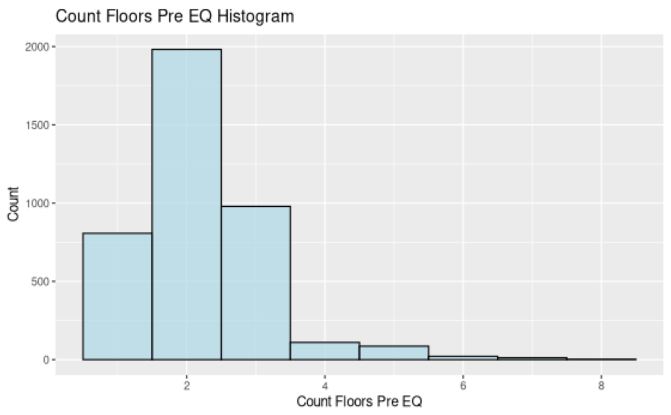

Fig. 4. Building floors histogram

When it comes to the categorical features, we found that there was a lot of redundant data, for example, *plan_configuration* or *legal_ownership_status* had a value distribution that was mostly the same, with one value being the

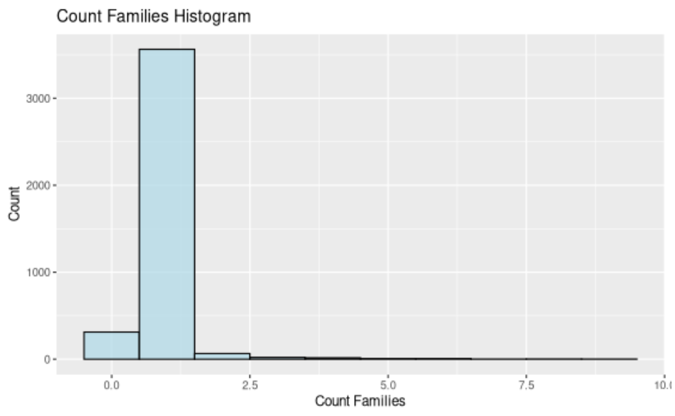

Fig. 5. Building family count histogram

majority, and the set of *secondary_use_X* also had a low value distribution, with most not having any secondary use, which does fit with our hypothesis of most buildings being residential houses. On the same line of categorical features we tried to extract any relation or meaning behind *surface_condition* and *position*, but we couldn't find any, as the values were mostly single characters and thus not very explanatory and there did not seem to be much correlation at first sight.

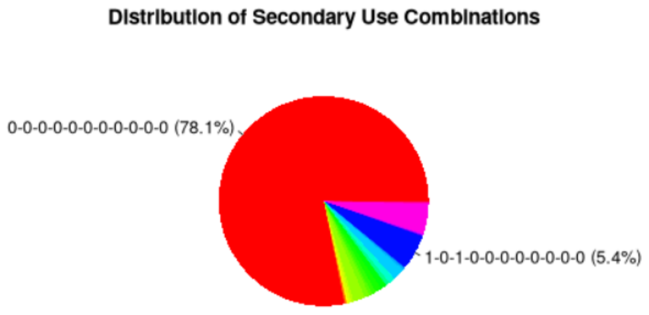

Fig. 6. Buildings secondary use data distribution

1) Data Correlation: We also conducted a linear correlation analysis to identify relationships between numerical features and the target variable, damage_grade, and found the following features presented the highest correlation values:

- *age* (0.269): We can infer that the age of the building is a significant factor in the damage grade, which is expected as older buildings are more likely to be damaged.
- *area_percentage* (-0.325): Larger area percentages are correlated with lower damage grades, which could mean that larger buildings might experience less relative damage.
- *roof_type* (-0.324): We can infer that the roof type is a significant factor in the damage grade, which is expected as some roof types are more resistant to earthquakes.
- *has_superstructure_mud_mortar_stone* (0.393), *has_superstructure_cement_mortar_brick* (-0.415), *has_superstructure_rc_non_engineered* (-0.221) and *has_superstructure_rc_engineered*(-0.259): We can infer that the superstructure material is a significant factor in the damage grade, which is expected as some materials are more resistant to earthquakes.

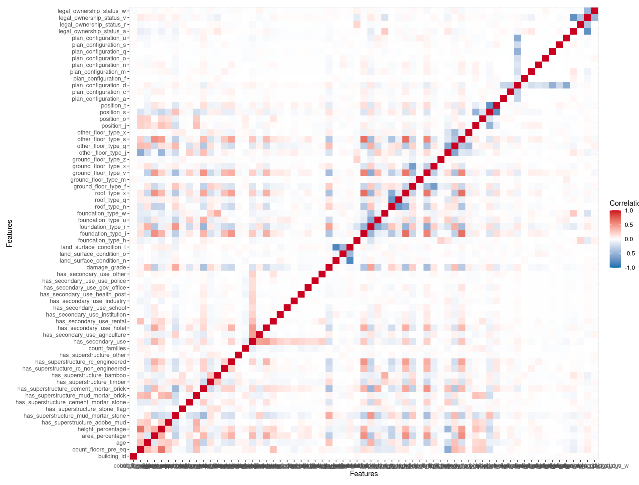

Fig. 7. Data correlation

2) Futher Analysis: Before taking any conclusions on the state of the data, we decided to carry out an analysis using the *DataExplorer* library, which report can be found alongside this document. The report did not shed light on anything that we hadn't already seen, but it served to reassure that the exploratory analysis we had done was correct. Overall the data seemed to be quite consistent and clean, although containing close to no variation in some columns, which could be a problem for the predictive model. And within some of those columns we decided some were best to be removed or processed in an specific way to reduce data complexity and potentially improve the model.

3) Proposed Preprocessing: After conducting the exploratory analysis, we proposed the following:

- Considering one hot encoding for the categorical features.
- Applying techniques to deal with the imbalanced data on the target class *damage_grade*.
- Removing seemingly redundant columns to reduce dimensionality and noise in the data.
- Potentially combining the has_secondary_use columns into a single column to simplify the data.

When it comes to removing columns, we propose a few that could removed if needed to reduce dimensionality or noise in the data, as we believe it wouldn't affect much the results. Taking in mind of course that building preliminary models to confirm this would be beneficial still:

- *building_id*
- *plan_configuration*
- *legal_ownership_status*
- *count_families*
- *has_secondary_use_use_police*
- *has_secondary_use_gov_office*
- *has_secondary_use_health_post*

- *has_secondary_use_industry*
- *has_secondary_use_institution*
- *has_secondary_use_school*
- *has_secondary_use_other*

C. Preprocessing

This part is explained in detail in `02. Preprocessing.Rmd`

We undertook a structured preprocessing phase to prepare the dataset for effective machine learning modeling. This phase focused on cleaning the data, addressing class imbalances, and formatting the dataset to optimize training and evaluation. Our process was divided into the following steps:

1) Data Cleaning: We began by ensuring the dataset's integrity and consistency. Following the insights gained during the EDA, we reduced the number of columns, getting rid of non-contributive features. The following steps were taken to clean the data:

- **Column Unification:** We consolidated related columns under the *has_secondary_use* group into a single column. This approach addressed inconsistencies and centralized the information for easier interpretation.
- **Column Elimination:** Based on our earlier analysis, we decided to eliminate columns such as:
  - *building_id*
  - *plan_configuration*
  - *legal_ownership_status*
  - *count_families*
  - *has_secondary_use_police*
  - *has_secondary_use_gov_office*
  - *has_secondary_use_health_post*
  - *has_secondary_use_industry*
  - *has_secondary_use_institution*
  - *has_secondary_use_school* and
  - *has_secondary_use_other*

  By doing so, we reduced dimensionality and noise, leaving 26 columns in the dataset.

2) Train-Test Split: To ensure a robust evaluation of our models, we split the dataset into training and testing subsets. We used a stratified approach to maintain the distribution of the target variable across both subsets. Using the *caret* library, we implemented this split to ensure a clear and reproducible methodology. We opted for an 80-20 split, with 80% of the data allocated for training and 20% for testing. This division allowed us to train the models on a substantial portion of the data while retaining a separate set for evaluation.

3) One-hot Encoding: We transformed categorical variables into numerical format using one-hot encoding. This step involved creating binary columns for categories within the features *land_surface_condition*, *foundation_type*, *roof_type*, *ground_floor_type*, *other_floor_type*, and *position*. While this increased the number of columns, it made the data compatible with machine learning algorithms. We saved the one-hot encoded dataset as *onehot* for future analysis.

4) Correlation Analysis: To address multicollinearity, we performed a correlation analysis. We identified two pairs of highly correlated columns: *land_surface_condition_n* and *position_s*, and *land_surface_condition_t* and *position_t*. In each pair, we retained the column with the higher variance to preserve the most informative features. We named the resulting dataset *filtered_onehot* and saved it for further analysis.

5) Principal Component Analysis (PCA): Next, we applied PCA to reduce the dataset's dimensionality. After standardizing the data to ensure consistent scaling, we used the Kaiser criterion to retain 13 principal components. These components captured significant variance while simplifying the dataset. Once PCA was completed, we reintroduced the target variable, *damage_grade*, and saved this version as *pca* for future steps.

6) Handling Imbalanced Data: Addressing class imbalance in the target variable was a priority. We adopted two different techniques to improve the dataset's balance:

- **SMOTE (Synthetic Minority Oversampling Technique):** We generated synthetic samples for the minority class using SMOTE. By creating new instances based on the nearest neighbors, we enhanced the representation of the minority class, enabling the model to learn effectively from all classes. We carried out the implementation using the *smotefamily* package in R, ensuring practical applicability and reproducibility. This balanced dataset was saved as *smote*.
- **Weight Allocation:** We applied another technique to mitigate the effects of class imbalance. We assigned weights to samples based on their class distribution. This method prioritized minority class instances during training, reducing prediction bias and improving overall performance. We saved this weighted dataset as *weighted* for subsequent analysis.

III. Analysis I

A. Supervised Analysis

This part is explained in detail in `03.02. Supervised Learning.Rmd`

The supervised analysis involved training and evaluating multiple machine learning models to predict the damage grade of buildings after an earthquake. We used the preprocessed datasets (*onehot*, *filtered_onehot*, *pca*, *smote*, and *weighted*) to compare the performance of various algorithms and preprocessing techniques.

Specifically, we decided to apply the following algorithms, available in the caret package:

- **Random Forest (RF):** A robust ensemble learning method that combines multiple decision trees to improve predictive accuracy and generalization. Random Forest is well-suited for classification tasks and can handle high-dimensional data effectively. In particular, we used the ranger implementation of Random Forest, which offers enhanced performance.
- **Gradient Boosting Machine (GBM):** An iterative ensemble method that builds decision trees sequentially, focusing on instances with higher prediction errors. GBM is known for its high predictive power and ability to capture complex relationships in the data.
- **Stochastic Gradient Boosting (SGB):** A variant of Gradient Boosting that introduces randomness into the algorithm, improving generalization and reducing overfitting. SGB is particularly effective for large datasets and high-dimensional feature spaces.

It is important to note that we also applied hyperparameter tuning to optimize the models' performance. The *caret* package provided a convenient framework for tuning the algorithms and selecting the best hyperparameters through grid search and cross-validation. By tuning the models, we aimed to improve their predictive accuracy and generalization capabilities, ensuring robust performance on unseen data. As a result, we generated 15 different models from the combination of the three algorithms and five datasets.

Another key aspect of the supervised analysis was evaluating the models' performance using appropriate metrics. We focused on the following evaluation metrics to assess the models' effectiveness:

- **Confusion Matrix:** As the target variable is categorical, we have focused on using this matrix and its metrics as it provides a lot of information about the model. In addition, we decided to use the extended version (*mode = "everything"*) that provides additional information such as the F1-Score, which is useful to us as it is the evaluation metric used in the ml-olympiad.
- **Area Under the Curve (AUC):** This metric is particularly useful for binary classification problems, providing insights into the model's ability to distinguish between classes. A higher AUC-ROC score indicates better performance, with a value of 0.5 representing random guessing and 1 representing perfect classification. In our case, we saw that R allowed the use of a multi-class auc function and initially used it. However, after the follow-up class, we read the paper [1] in which the implementation is discussed and discovered that in 50% of the cases, the results provided are incorrect. Therefore, following the professor's recommendation, we made use of the Python library defined in *Classification performance assessment for imbalanced multiclass data* [2], as it obtained really good results compared to the previous one, and we implemented a python wrapper in R using the *reticulate* library to use this python library (*imcp*).

1) Implementation: To reduce the code duplication and improve the readability of the supervised training, we implemented 3 functions to train the models, generate the confusion matrix and calculate the AUC-ROC score given some information like the *algorithm*, the *tuning grid*, the *dataset*, the *location* to store the predictions, the *test* dataset, etc.

2) In-depth analysis selection: Since we obtained 15 models as a result of the training, we have decided to choose 6 in order to comment on the most interesting results. In choosing these models, we have relied on the models with the best precision, recall, F1-Score and AUC; resulting in:

- **Best precision:** *rf_filtered* and *rf_weighted*
- **Best recall:** *rf_weighted* and *gbm_pca*
- **Best f1 (mean):** *rf_onehot* and *gbm_smote*
- **Best AUC:** *gbm_filtered* and *gbm_smote*

In the section IV-A we will analyse the results provided by the *rf_onehot*, *rf_filtered*, *rf_weigthed*, *gbm_filtered*, *gbm_pca* and *gbm_smote* models.

B. Unsupervised Analysis

This part is explained in detail in `03.01. Unsupervised Learning.Rmd`

The unsupervised analysis focused on exploring the dataset's structure, patterns, and relationships without using the target variable. We applied clustering techniques to identify natural groupings within the data and dimensionality reduction methods to visualize the dataset in lower dimensions.

Specifically, we used the following unsupervised learning techniques:

- **K-Means Clustering:** A popular clustering algorithm that partitions the dataset into k clusters based on the similarity of instances based on a distance metric. K-Means is effective for identifying distinct groups within the data and can help reveal underlying patterns or relationships.
- **Hierarchical Clustering:** A method that builds a hierarchy of clusters by recursively merging or splitting groups of instances. Hierarchical clustering provides insights into the data's structure and can be visualized using dendrograms to illustrate the relationships between clusters.
- **Autoencoder:** A neural network architecture used for dimensionality reduction and feature learning. Autoencoders learn to encode the input data into a lower-dimensional representation and then decode it back to the original space. By compressing the data into a latent space, autoencoders can capture essential features and patterns in the data.
- **t-SNE (t-Distributed Stochastic Neighbor Embedding):** A nonlinear dimensionality reduction

technique that maps high-dimensional data to a lower-dimensional space while preserving local relationships. t-SNE is particularly effective for visualizing complex datasets and revealing clusters or patterns that may not be apparent in higher dimensions.

We have done clustering on the *numeric* dataset, and we have used the *onehot* dataset for the dimensionality reduction techniques. We have used the *pca* dataset to compare the results of the clustering and dimensionality reduction techniques.

For evaluating optimal clustering number with *k-means* we have used the *elbow* and *silhouette* methods. Also we used *paarcord* to visualize high-dimensional data after clustering.

In order to compare the results of the dimensionality reduction we have compared 3 dimensionality reduction techniques: *pca*, *autoencoder* and *t-sne*. We have used the 3 first components of the *pca* dataset, for comparing the results of the reduction into 3 dimensions.

We have implemented the unsupervised analysis using the *cluster*, *dbscan* libraries for clustering, the *keras* library for the autoencoder and the *Rtsne* library for the t-SNE. We have also used the *factoextra*, *mass* and *plotly* libraries for the visualization.

1) Implementation: We have implemented the unsupervised analysis using the *cluster*, *dbscan* libraries for clustering, the *keras* library for the autoencoder and the *Rtsne* library for the t-SNE. We have also used the *factoextra*, *mass* and *plotly* libraries for the visualization.

2) In-depth analysis: We first applied the *k-means* and *hierarchical* clustering algorithms with the *numeric* dataset. Having no clear conclusions or patterns identified after the clustering, we decided follow up with dimensionality reduction techniques. We have applied the *autoencoder* to reduce the onehot dataset from 43 to 8 dimensions with minimal loss of information.

We have also applied an *autoencoder* to reduce it to 3 dimensions, and we have compared the results with the *pca* and *t-sne* techniques. We tried to visualize if there was any pattern in the data related to the target variable *damage_grade* or any pattern in building properties at all. For t-sne no clear pattern was found, but for *pca* and *autoencoder* we found that buildings with *damage_grade* 1 were mostly separated from *damage_grade* 2 and 3.

Then, we run the *k-means* clustering algorithms with the *autoencoder*, *pca* and *t-sne* datasets. We have used the elbow and silhouette methods to find the optimal number of clusters. We have found that the optimal number of clusters is 2 for the *autoencoder* and *pca* datasets, and 11 for the *t-sne* dataset. For the *pca* dataset we can see that the clusters kind of separate the buildings for *damage_grade* 1 from the rest, but for the *autoencoder* dataset the clusters are not as clear. For the *t-sne* dataset we can see that the clusters are not related to the *damage_grade* variable, giving us the idea that the *t-sne* is giving us patterns that are not related to the target variable.

Lastly, we ran *db-scan* clustering algorithm with the *autoencoder* and *t-sne* datasets. We have used the *elbow* methods to find the optimal *eps* value. We have found no clear clusters for the *autoencoder* dataset, but for the *t-sne* dataset we have found 14 clusters that are not related to the *damage_grade* variable.

The results of the unsupervised analysis are presented in section IV-B.

IV. Analysis II

A. Supervised Analysis Results

In this section, we present the results of the supervised analysis, focusing on the performance of the machine learning models across different datasets and algorithms. We will present the six best models based on the metrics we have mentioned above: precision, recall, F1 Score, and AUC for multi-class.

1) Precision: Precision measures the proportion of instances predicted as belonging to a particular class that actually belong to that class. We have calculated the precision for each class and the average precision for the selected models. The table below summarizes the precision scores for the selected models, and marks the best score for each class in bold:

TABLE I
Precision

Model	1	2	3	Avg
rf_onehot	0.655	0.554	0.492	0.568
rf_filtered	**0.664**	0.548	**0.568**	**0.594**
rf_weighted	0.503	**0.567**	0.434	0.501
gbm_filtered	0.617	0.549	0.453	0.540
gbm_pca	0.000	0.518	0.070	0.197
gbm_smote	0.574	0.551	0.454	0.526

As we can see, the *rf_filtered* model performs best overall, achieving the highest average precision of 0.594. It shows strong performance across all three classes, particularly for class 1 (0.664) and class 3 (0.568), indicating that it effectively minimizes false positives in these categories. The *rf_onehot* model follows closely with an average precision of 0.568, also demonstrating consistent but slightly weaker performance. On the other hand, the *gbm_pca* model struggles significantly, with a precision of 0 for class 1 and only 0.070 for class 3, resulting in a very low average precision of 0.197. This suggests that *gbm_pca* fails to make reliable predictions for certain classes, which means it may not be suitable for this earthquake damage prediction task. Other models like *rf_weighted*, *gbm_filtered*, and *gbm_smote* achieve average results, but they do not stand out in terms of precision.

2) Recall: Recall measures the proportion of instances of a particular class that are correctly identified by the model. We have calculated the recall for each class and the average recall for the selected models. The table below summarizes the recall scores for the selected models, and marks the best score for each class in bold:

As we can infer from the table, *rf_weighted* model stands out with the highest average recall of 0.558. This indicates

TABLE II
RECALL

Model	1	2	3	Avg
rf_onehot	0.548	0.760	0.250	0.519
rf_filtered	0.526	0.879	0.096	0.500
rf_weighted	**0.719**	0.295	**0.662**	**0.558**
gbm_filtered	0.526	0.671	0.331	0.509
gbm_pca	0.000	**0.884**	0.019	0.301
gbm_smote	0.519	0.584	0.435	0.513

its strong ability to correctly identify instances across all classes, particularly for class 1 (0.719) and class 3 (0.662). However, it underperforms for class 2, where its recall is only 0.295, showing some inconsistency in recovering instances of this class. Meanwhile, *gbm_pca*, despite its poor precision, achieves the highest recall for class 2 (0.884), suggesting it is effective at capturing most true instances of this class but fails to generalize well to others, as evidenced by its near-zero recall for class 1. The *rf_filtered* model, although strong in precision, lags in recall with an average score of 0.500, particularly struggling with class 3 (0.096), where it fails to capture a significant proportion of true instances. Other models like *rf_onehot*, *gbm_filtered*, and *gbm_smote* achieve average scores, with no standout performance in terms of recall.

3) F1 Score: A higher F1-Score reflects a better balance between precision and recall, crucial for minimizing false positives and false negatives in critical applications. In the table below, we present the F1 scores of the selected models and the best performing model for each class marked in bold:

TABLE III
F1-SCORE

Model	1	2	3	Avg
rf_onehot	**0.597**	0.641	0.332	**0.523**
rf_filtered	0.587	**0.675**	0.164	0.475
rf_weighted	0.591	0.387	**0.423**	0.501
gbm_filtered	0.568	0.603	0.382	0.518
gbm_pca	NA	0.653	0.030	0.341
gbm_smote	0.545	0.567	0.444	0.519

The comparison of F1-Scores highlights that the *rf_onehot* model performs best overall, achieving the highest average score of 0.523 and excelling in *Class 1*, though its performance for *Class 3* (0.332) is relatively weaker due to the fact that as we have seen in general terms *Class 3* is the most difficult to distinguish from *Class 2*. The *rf_filtered* model outperforms others for *Class 2* with an F1-Score of 0.675, while *rf_weighted* is most effective for *Class 3* at 0.423, showing the need to take into account the distribution of the classes of the dataset. On the other hand, models like *gbm_pca* struggle, with NA for *Class 1* due to zero precision and recall, indicating a complete failure to detect that class.

4) AUC: The AUC-ROC score provides insights into the models' ability to distinguish between classes, with higher scores indicating better performance. The table below presents the AUC-ROC scores for the selected models:

TABLE IV
AREA UNDER THE CURVE ROC

Model	AUC Multiclass
rf_onehot	0.436
rf_filtered	0.432
rf_weighted	0.423
gbm_filtered	0.443
gbm_pca	0.342
gbm_smote	**0.444**

The AUC-ROC scores show that the *gbm_smote* model performs best, with an AUC of 0.444, closely followed by *gbm_filtered* at 0.443. The *rf_onehot* model achieves an AUC of 0.436, indicating moderate performance in distinguishing between classes. The other models, such as *rf_filtered* and *rf_weighted*, exhibit similar AUC scores, ranging from 0.423 to 0.432. These results suggest that the models can differentiate between classes, with *gbm_smote* demonstrating the highest discriminatory power.

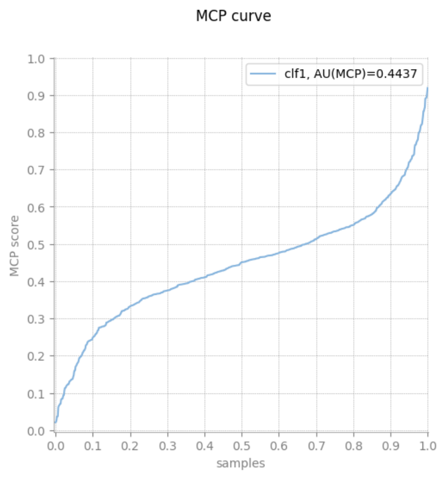

Fig. 8. AUC-ROC Curve for the *gbm_smote*

As shown in Figure 8, while the AUC is relatively close to 0.5, it suggests that our model has some discriminatory power, but it is not highly effective at separating the classes. The MCP curve rises steadily without sharp increases, indicating that our model's predictions are only moderately better than random guessing. This AUC score reflects the performance of the *gbm_smote* model as one of the best among the models we evaluated, though there is still room for improvement.

B. Unsupervised Analysis Results

1) K-Means Clustering: The K-Means clustering algorithm was applied to the *numeric* dataset to identify natural groupings within the data. We used the *elbow* and *silhouette* methods to determine the optimal number of clusters. The results are summarized below:

- **Elbow Method:** The elbow method did not give us any valuable insights on the number of clusters, as the plot did not show a clear inflection.

- **Silhouette Method:** The silhouette method suggested that the optimal number of clusters is 2, with a silhouette score of $\tilde{0}.25$.

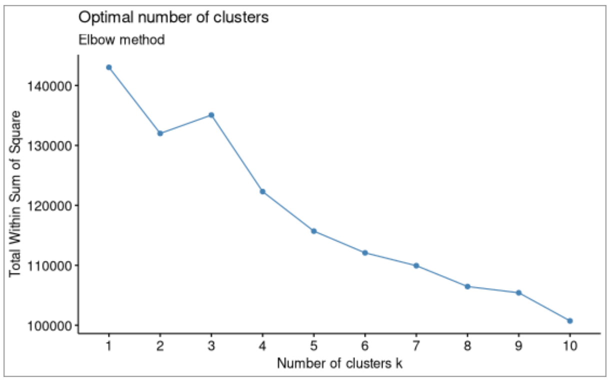

Fig. 9. Elbow method for K-means clustering

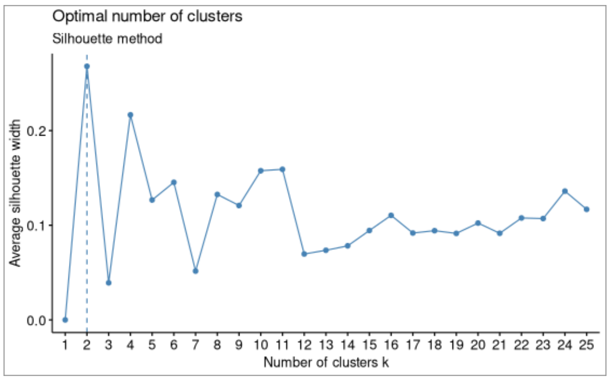

Fig. 10. Silhouette method for K-Means clustering

With this information (in Figures 9 and 10 we procceded to plot the data using the parallel coordinates plot, which allowed us to visualize the clusters and their distribution. The plot did not showed any clear difference, as most features were binary and those which weren't did not show any clear pattern or difference in value distribution.

2) Hierarchical Clustering: With this in mind we decided to look for finer-grained clusters using the hierarchical clustering algorithm, with the goal of finding more detailed patterns in the data. We used two methods: *ward.D2* and *complete*, and plotted the dendrograms for both. The results were very close to non-informative. Both finished with a very large cluster and a significantly smaller one and there did not seem to be any patters on the data once more.

After this we decided to move on to the dimensionality reduction techniques as to extract patters with more complex techniques which may not be able to be extracted with traditional clustering techniques. These techniques would also allow us to visualize the data in lower dimensions, which would be useful for interpretation.

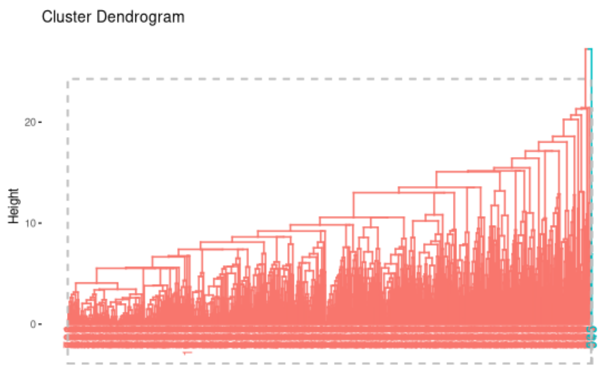

Fig. 11. Dendogram for *complete* hierarchical clustering method

3) Autoencoder: We applied the autoencoder to the *onehot* dataset to reduce the dimensions from 43 to 8. The autoencoder was trained for 100 epochs, and the loss decreased steadily over time, indicating that the model was learning the underlying structure of the data. As we obtained very promising results, we decided to apply again the autoencoder to reduce the dimensions to 3, and then plot it in three dimensions. The results showed two clearly distinct clusters, which after colouring with the data of the *damage_grade* column, showed one of the clusters mostly corresponding to buildings with *damage_grade* equal to 1, and the other group being more noisy with no clear differentiation of properties between building with *damage_grade* equal to 2 and 3.

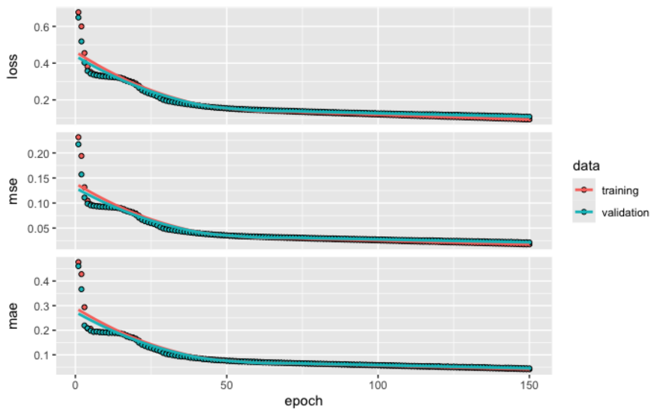

Fig. 12. Autoencoder performance during training

4) t-SNE: After that, we tried with the t-SNE technique, which is a non-linear dimensionality reduction technique that maps high-dimensional data to a lower-dimensional space while preserving local relationships. We applied t-SNE to the *onehot* dataset to reduce the dimensions to 3 and visualize the data in a lower-dimensional space. The results showed a more complex distribution of the data, but this time it did show clear separation between clusters, which were not related to the *damage_grade* variable.

5) K-Means over 3 dimensions: After dimensionality reduction, we applied the K-Means clustering algorithm to the datasets generated by the autoencoder, t-Sne and PCA

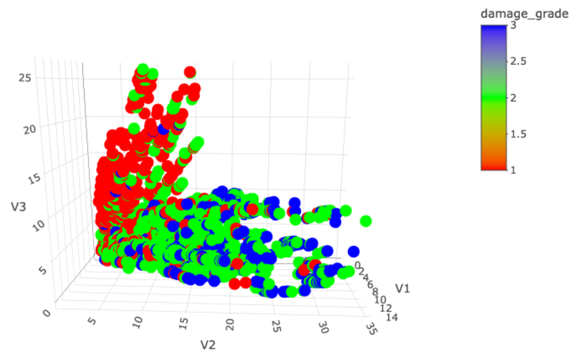

Fig. 13. Encoded data plotted in 3D

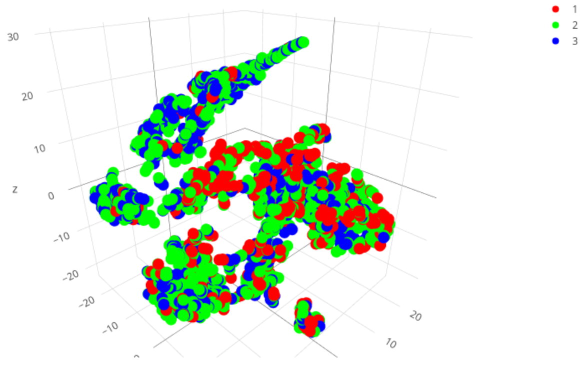

Fig. 14. Tsne data plotted in 3D

techniques. We again applied the elbow and silhouette methods to determine the optimal number of clusters for each dataset. Finding once more for the autoencoder and PCA datasets that the optimal number of clusters was 2, and for the t-SNE dataset we found the optimal number to be 11, which reafirms the idea that the t-SNE is giving us patterns that are not related to the target variable. We will discuss these implications further in the document.

6) DB-Scan: Lastly, we applied the DB-Scan clustering algorithm to the autoencoder and t-SNE datasets. We used the elbow method to find the optimal *eps* value for each dataset. We found that for the autoencoder dataset there were no clear clusters, but for the t-SNE dataset we found 13 clusters and several outliers.

V. Discussion

The data presented challenges related to its interpretability, variability, and balance, which significantly impacted the predictive modeling results. A critical limitation was the low interpretability of categorical features, such as those encoded with single characters, which hindered the ability to extract meaningful patterns and relationships. Furthermore, low variability in some features limited the model's capacity to generalize and differentiate between the damage grades

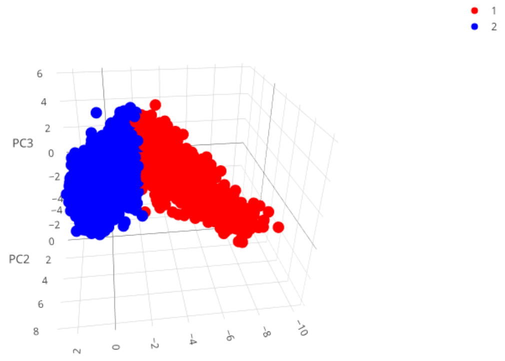

Fig. 15. Clusters for the PCA dataset (first 3 dimensions)

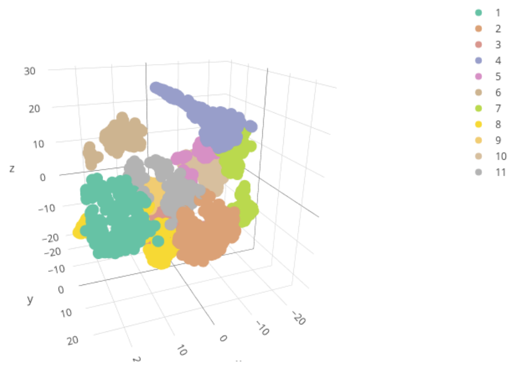

Fig. 16. t-Sne clusters obtained with K-Means

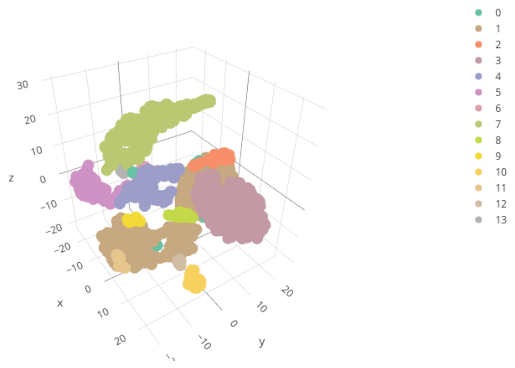

Fig. 17. t-Sne clusters obtained with DB-Scan

effectively. Combined with an unbalanced class distribution, the models struggled to achieve consistent accuracy across all categories, particularly for damage grades 2 and 3.

Dimensionality reduction techniques, including PCA, autoencoders, and t-SNE, were explored to identify latent patterns and simplify data representation. While these methods revealed clusters, especially through t-SNE, these clusters were not strongly correlated with the target variable, suggesting a lack of granularity or completeness in the dataset. For instance, crucial data points such as the distance from the earthquake epicenter, adjacent structures, and other environmental factors were missing, limiting the depth of analysis. Additionally, the arbitrary selection of damage grade 2 and 3 instances, potentially to increase challenge complexity in the ML Olympiad, may have introduced random noise, further reducing predictive accuracy.

VI. CONCLUSIONS

This study underscores significant limitations and insights into the dataset and its influence on modeling earthquake damage to buildings. The data's low interpretability and variability, coupled with its unbalanced distribution, created challenges for extracting meaningful relationships, particularly between damage grades 2 and 3. Despite employing advanced preprocessing and analysis techniques, the lack of distinctive patterns between these grades limited the models' predictive effectiveness. Dimensionality reduction and clustering methods offered some insight but failed to provide robust segmentation tied to the target variables.

Future improvements should focus on collecting and integrating additional, highly relevant features, such as geographical data, surrounding building characteristics, and other environmental attributes, to enhance the granularity of the dataset. Furthermore, a detailed review of the data preparation process to avoid arbitrary selection biases could yield more representative distributions. While current findings highlight the limitations, they also pave the way for more robust, feature-rich datasets that can better predict building resilience in seismic events.

VII. OUTSTANDING TASKS

A. Kaiser Criterion

The Kaiser criterion is a widely used method for determining the number of principal components to retain during PCA. It suggests retaining components with eigenvalues greater than 1, as these components explain more variance than a single original variable. By selecting components with eigenvalues above this threshold, we can capture the most significant information in the data while reducing dimensionality. The Kaiser criterion is a valuable tool for balancing the trade-off between complexity and information retention in PCA, ensuring that the transformed dataset remains informative and interpretable.

B. SMOTE

SMOTE (Synthetic Minority Oversampling Technique) is a powerful method for addressing class imbalance in machine learning datasets. It generates synthetic samples by interpolating between a minority class sample and its nearest neighbors. This technique enhances the representation of underrepresented classes, enabling models to learn effectively from all classes.

Mathematically, SMOTE selects a minority class sample x and one of its k-nearest neighbors x_{neighbor}, identified using a distance metric like Euclidean distance. It then creates a synthetic point $x_{\text{synthetic}}$ using the formula:

$$x_{\text{synthetic}} = x + \lambda \cdot (x_{\text{neighbor}} - x)$$

Here, λ is a random number between 0 and 1, ensuring the new sample lies randomly along the line connecting x and x_{neighbor}. This approach increases the density of the minority class in feature space, providing a more nuanced representation compared to simple duplication.

SMOTE is particularly useful for classification tasks where one class is significantly smaller than the others, as it helps prevent bias and improves the model's ability to generalize to unseen data. While effective, SMOTE should be used carefully, as oversampling can introduce noise and affect the model's performance, or even lead to overfitting.

By incorporating SMOTE into the preprocessing pipeline, we can create a more balanced dataset that leads to better model performance and robust predictions.

C. AUC multiclass wrapper

This code leverages two powerful libraries to bridge R and Python functionalities:

1) **`reticulate` (R Library)**:
    - The `reticulate` library in R provides seamless integration between R and Python. It allows users to run Python scripts, exchange data between R and Python, and access Python libraries directly from R. In our implementation, `reticulate` is used to execute a Python script (`auc.py`) that performs AUC (Area Under Curve) calculations and generates a plot.
2) **`imcp` (Python Library)**:
    - The `imcp` Python library is a specialized tool for calculating the Integrated Multi-class Performance Curve (IMCP) and its associated area. It provides utilities like `plot_mcp_curve` for visualization and `imcp_score` for performance scoring. These functions are used to calculate the AUC and create a plot based on the provided data.

1) Implementation Details:

R Function Overview (`get_AUC_and_plot` in `03.02.Supervised Learning.Rmd`): This function serves as a wrapper to preprocess data in R, execute a Python script, and return results back to R. Key steps include:

- **Preprocessing**: Subsetting the predicted data to include only relevant columns and saving both original and predicted datasets as temporary CSV files.
- **Python Execution**: Using `py_run_string` and `py_run_file` from `reticulate` to execute the

Python script (`auc.py`), which calculates the AUC and saves a performance curve plot.

- **Return Values**: The computed AUC value is retrieved from Python and returned alongside the plot path.

Python Script Overview (`auc.py` in `/scripts/auc.py`): This script is responsible for the core computation and visualization:

- **Data Loading**: The script reads input data files containing ground truth (`original_data`) and predictions (`predicted_data`).
- **AUC Calculation**: The `imcp_score` function calculates the AUC based on the IMCP framework, offering a detailed multi-class performance metric.
- **Plot Generation**: The `plot_mcp_curve` function generates a visualization of the performance curve, which is saved as an image for reporting purposes.

Data Flow Between R and Python: The R function saves data as CSV files, which act as an interface between the two environments. These files are read by the Python script for analysis. Similarly, the Python script outputs a saved plot and the AUC score, which R retrieves and integrates into its workflow.

D. Autoencoder

The autoencoder is a type of artificial neural network used for unsupervised learning, particularly for dimensionality reduction. It consists of two main parts: the encoder, which compresses the input data into a lower-dimensional representation, and the decoder, which reconstructs the original data from this compressed form. This technique is valuable because it allows for the extraction of meaningful features from high-dimensional data while minimizing information loss.

In our analysis, we applied the autoencoder to the onehot dataset, reducing its dimensionality from 43 to 8. This reduction was achieved with minimal loss of information, making it easier to visualize and analyze the data. Furthermore, we explored reducing the dataset to 3 dimensions, which facilitated the application of clustering algorithms. The results indicated that the autoencoder effectively captured the underlying structure of the data, revealing clusters that corresponded to different damage grades.

E. t-SNE

t-SNE (t-distributed Stochastic Neighbor Embedding) is a powerful non-linear dimensionality reduction technique particularly suited for visualizing high-dimensional datasets. It works by converting similarities between data points into joint probabilities and then minimizing the divergence between these probabilities in the high-dimensional space and the lower-dimensional representation. This method is especially valuable for visualizing complex datasets, as it preserves local structures and reveals clusters that may not be apparent in the original high-dimensional space.

In our study, we applied t-SNE to the onehot dataset to reduce its dimensions to 3, allowing for effective visualization of the data. The resulting plots demonstrated a complex distribution with several clusters; however, these clusters did not correlate with the `damage_grade` variable contrary to the other techniques applied. This finding suggests that t-SNE is to be applied in different contexts compared to other dimensionality reduction techniques and may be used in parallel to extract additional insights from the data.

References

[1] D. J. Hand and R. J. Till, ”A simple generalisation of the area under the ROC curve for multiple class classification problems,” *Machine Learning*, vol. 45, pp. 171–186, 2001.

[2] J. S. Aguilar-Ruiz and M. Michalak, ”Classification performance assessment for imbalanced multiclass data,” *Scientific Reports*, vol. 14, no. 1, pp. 10759, 2024.