Predictive Modeling of Building Damage in Earthquakes with Machine Learning

Bharath Chandra Beeravelly
Department of Computer Science
Stevens Institute of Technology
Hoboken, NJ, USA
bbeerave@stevens.edu

Bhavya Shree Arcot Murugeshan
Department of Mathematical Sciences
Stevens Institute of Technology
Hoboken, NJ, USA
barcotmu@stevens.edu

Pranjali Rajeshwar Andhale
Department of Mathematical Sciences
Stevens Institute of Technology
Hoboken, NJ, USA
pandhale@stevens.edu

***Abstract*—Earthquakes can cause devastating damage to buildings, resulting in far-reaching consequences for human lives, social structures, and geographical landscapes. In this paper, we present a machine learning approach to predict the damage grade of buildings impacted by earthquakes. We explore various machine learning algorithms for predicting and classifying damage grades and evaluate their performance. Our approach has the potential to make a significant contribution to the field of earthquake damage prediction.**

I. Introduction

The devastating effects of earthquakes on buildings have far-reaching consequences on human lives, social structures, and geographical landscapes. Earthquakes can displace communities and populations, straining essential resources. As a result, it is crucial to predict the extent of building damage to develop effective prevention and mitigation strategies. In this project, we aim to predict the damage grade of buildings impacted by earthquakes, focusing on the Gorkha earthquake that struck Nepal in 2015. Our methodology involves exploring various machine learning algorithms to predict and classify the damage grades of buildings. We preprocess the data, handle missing values, and encode categorical values. We then split the dataset into training and testing subsets and apply different classification models, such as Logistic Regression, Random Forests, Neural Networks, and XGBoost, to evaluate their performance in predicting damage grades. We measure model performance using the micro-averaged F1 score. By accurately predicting building damage grades, we aim to provide valuable insights for urban planners, engineers, and disaster management agencies to develop better policies, construction guidelines, and early warning systems to minimize the loss of life and property during earthquakes.

II. Related Work

Our project is based on an online challenge titled Richter's Predictor: Modeling Earthquake Damage, hosted by DrivenData, a data science competition platform that hosts online challenges for social impact. This competition aimed to develop a machine learning model to predict damage levels for buildings affected by the 2015 Nepal earthquake, using a dataset provided by the organizers. Although no previous work has been conducted on this specific problem, there has been considerable research on earthquake damage prediction using machine learning techniques. Existing literature on similar problems and related techniques may inform our approach. For example, Wang et al. (2019)[1] used a convolutional neural network (CNN) to analyze post-earthquake images and predict the degree of building damage caused by earthquakes in China. Huang et al. (2020)[2] used a deep neural network (DNN) approach to predict peak ground acceleration of earthquakes, which can be used to estimate building damage. Chong et al. (2019)[3] used a support vector machine (SVM) algorithm to predict the degree of damage to buildings caused by earthquakes, using building and seismic data from the 2016 Kumamoto earthquake in Japan. Tymvios et al. (2020)[4] used a random forest algorithm to predict the probability of building damage in Cyprus due to earthquakes, using building, geological, and seismic data. Overall, the existing literature suggests that machine learning algorithms, including CNNs, DNNs, SVMs, and random forests, can be effective in predicting earthquake damage levels. These studies demonstrate the potential for these models to be applied in different contexts and regions.

III. Our Solution

A. Description of Dataset

The dataset comprises information on buildings impacted by the Gorkha Earthquake. Each row of the dataset represents a specific building in the region, and the dataset contains 39 columns. The building id column is a unique and random identifier, while the remaining 38 features describe the building's structure and legal ownership. Categorical variables have been obfuscated using random lowercase ASCII characters. Numerical data has been scaled, and categorical data has been label-encoded. However, for certain nominal categories, we use one-hot encoding instead of label

B. *Machine Learning Algorithms*

To tackle the classification problem, we have implemented various machine learning algorithms, including Multi-class Logistic Regression, Random Forest, XGBoost, and Neural Networks. Although Logistic Regression is a fundamental algorithm, we have employed it to assess its performance in this context. Random Forest and XGBoost are cutting-edge classification models, and we have utilized them to compare their performance against other models. Additionally, we have implemented a basic Neural Network with two hidden layers, consisting of 64 and 32 units, respectively.

C. *Implementation Details*

We utilize the featurewiz function from the Featurewiz library for automatic feature selection. Featurewiz identifies and selects the most important features based on their correlation with the target variable while considering multi-collinearity. We set the correlation limit to 0.7, meaning that any two features with a correlation greater than 0.7 will be considered highly correlated, and only one of them will be selected.

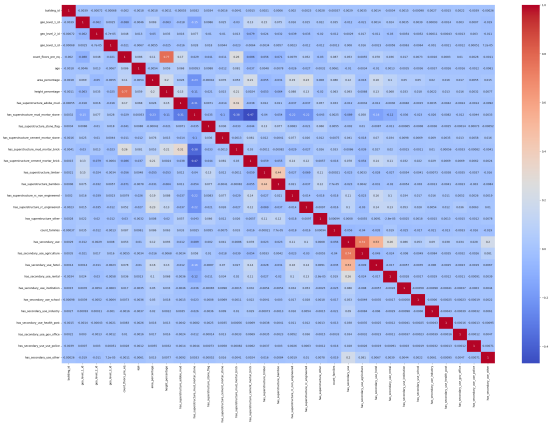


Fig. 1: Correlation Matrix

We then create new training, testing, and validation datasets containing only the most important features identified by featurewiz. This process reduces the dimensionality of the dataset, making our analysis more efficient and focused on relevant features. Using these important features in our machine learning models can improve the accuracy and performance of our predictions, ultimately aiding the development of effective earthquake damage mitigation strategies. We preprocess our data by removing the 'foundation_type' feature and transforming the 'age' feature with a logarithmic transformation (adding a constant of 0.1 to handle zero values). We then one-hot encode categorical features ('roof_type', 'position', 'ground_floor_type', and 'other_floor_type') and scale continuous features ('geo_level_1_id', 'count_floors_pre_eq', 'count_families', 'geo_level_2_id', and 'age_log') using MinMaxScaler.

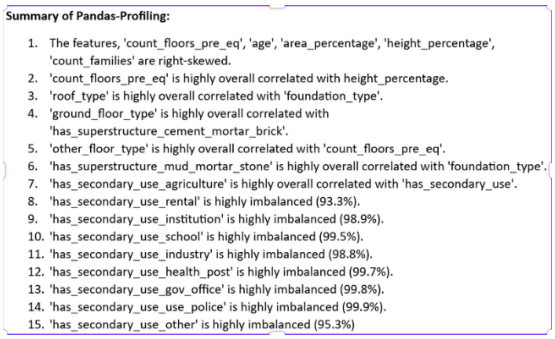


Fig. 2: Summary of Pandas Profiling

These preprocessing steps help improve the performance of our machine learning models by normalizing the distribution of features, reducing the impact of outliers, and enabling models to better interpret and utilize the information. To address class imbalance in our training dataset, we employ the Synthetic Minority Over-sampling Technique (SMOTE), which generates synthetic samples for the minority class by interpolating between existing instances. We also encode the categorical columns in the original unprocessed datasets (train, test, and validation) using LabelEncoder. These steps prepare the datasets for efficient and accurate analysis by our machine learning models.

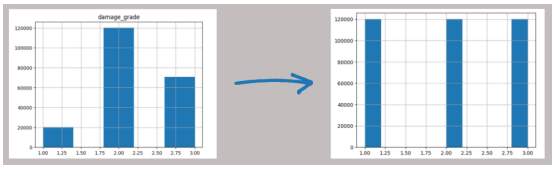


Fig. 3: Data Distribution

We applied Principal Component Analysis (PCA) to reduce the dimensionality of our dataset and capture the most important information. After fitting the PCA model with 30 components to the oversampled training dataset, we calculated the explained variance ratio and observed that around 8 principal components are sufficient to summarize the data effectively.

We decided to use 17 principal components to retain 95% of the explained variance. Then, we applied PCA with 17 components to the train, validation, and test datasets, obtaining the PCA-transformed datasets. Next, we employed the OneVsRestClassifier (OVRClassifier) algorithm from the sklearn.multiclass library. This approach trains a binary classifier for each class to predict whether an instance belongs to that class or not. The classifier with the highest predicted probability is used to assign the class label to the instance. By using PCA and the OVRClassifier, we aim to improve the efficiency and performance of our machine learning models in predicting earthquake damage grades for buildings. These techniques will contribute to providing valuable insights for urban planners,

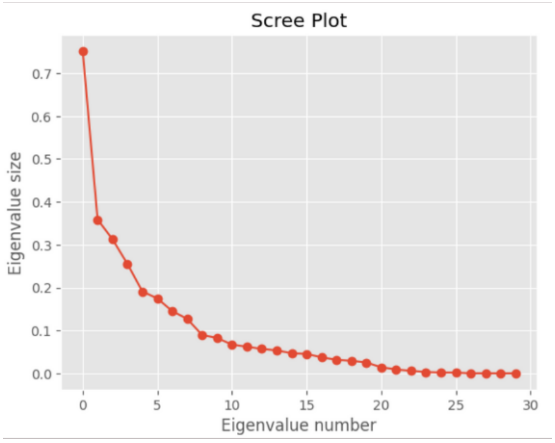


Fig. 4: Principal Component Analysis

engineers, and disaster management agencies to develop better policies, construction guidelines, and early warning systems to minimize the loss of life and property during earthquakes. We train and evaluate various machine learning models on our raw, preprocessed and selected features and pca trained data to predict earthquake damage grades for buildings: Decision Tree Classifier, Neural Network, Logistic Regression, Random Forest Classifier, XGBoost Classifier.

IV. **Comparison**

By comparing the performance of these models, we identify the best model with the highest accuracy and F1-score for predicting earthquake damage grades for buildings. This analysis enables urban planners, engineers, and disaster management agencies to develop better policies, construction guidelines, and early warning systems to minimize the loss of life and property during earthquakes.

The OvR Classifier, which stands for One-vs-Rest Classifier, is a popular approach in machine learning for multi-class classification problems. It involves training multiple binary classifiers, each representing one class against the rest of the classes. In this scenario, the OvR Classifier was initially implemented using the Logistic Regression algorithm as the base model.

Before performing a Grid Search, which is a technique used to optimize hyperparameters, the OvR Classifier had the following configuration:

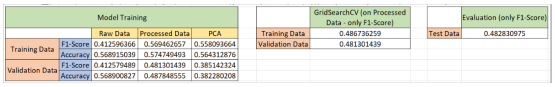
Base model: Logistic Regression Estimator max_iter: 100 Estimator penalty: l1 After applying the Grid Search, which explores different combinations of hyperparameters to find the best configuration, the OvR Classifier was modified with the following changes:

Base Model: SVM (Support Vector Machine) Estimator max_iter: 500 Estimator penalty: l2 The base model refers to the underlying algorithm used for classification, and in this case, it was changed from Logistic Regression to SVM. SVM is a powerful algorithm known for its effectiveness in handling complex decision boundaries. It attempts to find the optimal hyperplane that separates the classes in the feature space.

Additionally, the hyperparameters of the estimator were adjusted. The max_iter parameter was increased from 100 to 500. This parameter determines the maximum number of iterations the algorithm can take to converge. Increasing it allows for potentially better convergence and learning.

The penalty parameter was also modified from l1 to l2. This parameter influences the type of regularization applied to the model's weights during training. L1 regularization encourages sparsity by shrinking less important feature weights to zero, while L2 regularization penalizes large weights. Switching from l1 to l2 suggests a change in the emphasis of the model's regularization strategy.

Overall, these changes indicate a refinement in the OvR Classifier. The switch to SVM as the base model and the adjustments in hyperparameters suggest an exploration of different algorithms and parameter settings to improve the model's performance. By utilizing Grid Search, the OvR Classifier aims to find the optimal combination of these elements to enhance its classification accuracy and generalization ability.



Model Training				
		Raw Data	Processed Data	PCA
Training Data	F1-Score	0.412596366	0.56946265	0.558093664
	Accuracy	0.568915039	0.574749493	0.564312876
Validation Data	F1-Score	0.412579489	0.481301439	0.385142324
	Accuracy	0.568900827	0.487848555	0.382280208

	GridSearchCV (on Processed Data - only F1-Score)
Training Data	0.486736259
Validation Data	0.481301439

	Evaluation (only F1-Score)
Test Data	0.482830975

Fig. 5: OneVsRest Classifier

In the context of decision trees, the configuration of the model can significantly impact its performance and generalization ability. Here, we'll discuss the changes made to a decision tree classifier before and after applying Grid Search with Cross-Validation (GSCV) to optimize its hyperparameters.

Before performing GSCV, the decision tree classifier had the following configuration:

Class weight: None Criterion: 'gini' Max depth: None In decision trees, the class weight parameter determines the importance given to each class during training. When set to None, all classes are considered equal in terms of importance. The criterion parameter specifies the quality measure used to evaluate the splits at each node, and in this case, 'gini' was chosen. 'Gini' refers to the Gini impurity, a metric that quantifies the level of impurity in a node's class distribution. Lastly, the max depth parameter was set to None, indicating that the decision tree could grow without any restrictions on the depth.

After applying GSCV, the decision tree classifier underwent the following modifications:

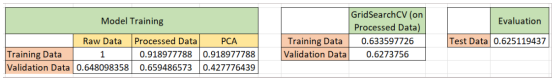
Class weight: Balanced Criterion: 'gini' Max depth: 10 The class weight parameter was changed to 'Balanced.' This setting automatically adjusts the weights inversely proportional to the class frequencies in the training data. It is particularly useful when dealing with imbalanced datasets, where some classes are underrepresented. By

assigning higher weights to minority classes, the decision tree can give them more importance during training, potentially leading to improved performance.

The criterion parameter remained unchanged, still using 'gini' as the impurity measure. 'Gini' impurity evaluates the quality of splits based on the probability of misclassifying a randomly chosen element in a given node.

The max depth parameter was set to 10. This parameter restricts the maximum depth of the decision tree. Limiting the depth can prevent overfitting, as deeper trees tend to capture more noise and may not generalize well to unseen data. By specifying a maximum depth of 10, the decision tree is constrained, potentially resulting in a more balanced trade-off between model complexity and generalization.

By utilizing GSCV, which systematically explores various combinations of hyperparameters, the decision tree classifier aims to find the optimal configuration that maximizes its classification performance. The adjustments made to the class weight, max depth, and the retention of the 'gini' criterion suggest a deliberate effort to address class imbalance and control the tree's depth, factors that can influence the decision tree's overall effectiveness in making accurate predictions.



Model Training			
	Raw Data	Processed Data	PCA
Training Data	1	0.918977788	0.918977788
Validation Data	0.648098358	0.659486573	0.427776439

	GridSearchCV (on Processed Data)
Training Data	0.633597726
Validation Data	0.6273756

	Evaluation
Test Data	0.625119437

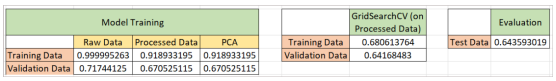
Fig. 6: Decision Tree

Random Forest is an ensemble learning method that combines multiple decision trees to improve prediction accuracy and reduce overfitting. Let's discuss the changes made to a Random Forest classifier before and after adjusting its hyperparameters.

Before the adjustments, the Random Forest classifier had the following configuration:

n_estimators: 100 max_depth: None max_features: None bootstrap: False class_weight: None The n_estimators parameter determines the number of decision trees in the random forest. In this case, it was set to 100, meaning the model consisted of 100 individual decision trees. The max_depth parameter was set to None, indicating that there were no constraints on the depth of the trees, allowing them to grow until all leaves were pure or until another stopping criterion was met. The max_features parameter was also set to None, meaning that the model considered all features for each split at every node. The bootstrap parameter was set to False, indicating that the entire dataset was used for training each tree. Lastly, the class_weight parameter was set to None, treating all classes equally during training.

After the adjustments were made, the Random Forest classifier had the following configuration:



Model Training			
	Raw Data	Processed Data	PCA
Training Data	0.999995263	0.918933195	0.918933195
Validation Data	0.71744125	0.670525115	0.670525115

	GridSearchCV (on Processed Data)
Training Data	0.680613764
Validation Data	0.64168483

	Evaluation
Test Data	0.643593019

Fig. 7: Random Forest

bootstrap: False max_depth: 15 max_features: 'auto' n_estimators: 100 The bootstrap parameter remained unchanged, still set to False. This means that the model continued to use the entire dataset for training each decision tree, without performing random sampling with replacement.

The max_depth parameter was set to 15, which restricts the maximum depth of each decision tree in the random forest. By limiting the depth, the model can prevent overfitting and promote better generalization to unseen data. The choice of 15 as the maximum depth value indicates a desire to control the complexity of the individual trees in the forest.

The max_features parameter was modified to 'auto', which means that the model automatically determines the number of features to consider for each split at every node. This selection is based on a heuristic, such as the square root of the total number of features. By using this setting, the model can introduce randomness and diversity among the decision trees, which can improve the overall performance and robustness of the random forest.

Lastly, the n_estimators parameter remained unchanged at 100, indicating that the random forest still consisted of 100 decision trees.

These adjustments in the hyperparameters of the Random Forest classifier were made to optimize its performance. The changes aim to control the depth of the trees, limit the number of features considered at each split, and maintain the number of decision trees in the forest. Additionally, the bootstrap parameter was retained, indicating that the model continued to use the entire dataset for training. These modifications can enhance the model's ability to capture complex patterns, reduce overfitting, and improve the accuracy of predictions.

XGBoost (Extreme Gradient Boosting) is a popular machine learning algorithm known for its efficiency and accuracy. Let's discuss the changes made to an XGBoost classifier before and after modifying its hyperparameters.

Before the modifications, the XGBoost classifier had the following configuration:

Booster: 'gblinear' Learning rate: 0.01 Max depth: 6 n_estimators: 100 Objective: binary:logistic The booster parameter determines the type of booster used in XGBoost. In this case, it was set to 'gblinear', which refers to the linear booster. The linear booster builds the model by applying linear functions to the features.

The learning_rate parameter controls the shrinkage applied to each tree. A lower learning rate results in slower learning but potentially better convergence. Here, it was

Model Training			
	Raw Data	Processed Data	PCA
Training Data	0.740779002	0.786972926	0.71124944
Validation Data	0.715369594	0.697045446	0.434417697

	GridSearchCV (on Processed Data)
Training Data	0.65241511
Validation Data	0.644935857

	Evaluation
Test Data	0.643634453

Fig. 8: XGBoost

set to 0.01, indicating a small step size for updating the model's weights.

The max_depth parameter determines the maximum depth of each tree in the ensemble. In the initial configuration, it was set to 6, which means each tree could grow up to a depth of 6 levels.

The n_estimators parameter specifies the number of trees to build. In this case, 100 trees were constructed.

The objective parameter defines the loss function to be optimized during training. 'binary:logistic' refers to the binary logistic loss, which is suitable for binary classification problems.

After the adjustments were made, the XGBoost classifier had the following configuration:

Booster: 'gbtree' Learning rate: 0.01 Max depth: 10 n_estimators: 100 Objective: binary:logistic The booster parameter was changed to 'gbtree'. This implies the use of gradient boosting with tree-based models. The 'gbtree' booster builds each tree iteratively, focusing on reducing the residuals or errors.

The learning_rate parameter remained unchanged at 0.01, indicating the same small step size for updating the model's weights.

The max_depth parameter was increased to 10. By allowing each tree to grow deeper, the model can capture more complex interactions and potentially improve its ability to learn intricate patterns in the data.

The n_estimators parameter remained at 100, implying the same number of trees were built as before.

The objective parameter remained as 'binary:logistic', indicating the use of the binary logistic loss function for binary classification tasks.

These modifications in the XGBoost hyperparameters aimed to explore different boosting strategies and adjust the model's capacity to capture more complex patterns. By switching to 'gbtree' as the booster and increasing the max_depth, the XGBoost classifier can potentially leverage the power of tree-based models to improve its predictive performance. The learning_rate and n_estimators parameters remained the same, suggesting that the adjustments primarily focused on the algorithm's inherent characteristics and capacity to handle complex data patterns.

Artificial Neural Networks (ANNs) are powerful models that mimic the structure and function of biological neural networks. Let's discuss the changes made to an ANN before and after modifying its architecture and optimizer.

Before the modifications, the ANN had the following configuration:

Hidden layer 1 neurons: 64 Hidden layer 2 neurons: 32 Activation functions: ['relu', 'relu', 'softmax'] Optimizer: 'Adam' The hidden layer 1 neurons parameter refers to the number of neurons in the first hidden layer, which was set to 64. Hidden layers are intermediate layers between the input and output layers, responsible for extracting and learning relevant features from the input data.

The hidden layer 2 neurons parameter was set to 32, indicating the number of neurons in the second hidden layer. The presence of multiple hidden layers allows the model to capture complex relationships and hierarchies in the data.

The activation functions parameter specified the activation functions used in each layer. In this case, 'relu' (Rectified Linear Unit) was chosen for both hidden layers, while 'softmax' was used in the output layer. 'relu' is a widely used activation function that introduces non-linearity into the model, enabling it to learn complex patterns. 'softmax' is commonly used for multi-class classification, as it produces probabilities for each class, ensuring the predicted class probabilities sum up to 1.

The optimizer parameter was set to 'Adam'. Adam (Adaptive Moment Estimation) is an optimization algorithm widely used in deep learning. It combines the benefits of both AdaGrad and RMSProp algorithms, adapting the learning rate on a per-parameter basis and efficiently handling sparse gradients.

After the adjustments were made, the ANN had the following configuration:

Hidden layer 1 neurons: 64 Hidden layer 2 neurons: 64 Activation functions: ['relu', 'relu', 'softmax'] Optimizer: 'RMSProp' The hidden layer 1 neurons remained the same at 64.

The hidden layer 2 neurons parameter was also set to 64, matching the number of neurons in the first hidden layer. By increasing the number of neurons in the second hidden layer, the model has the potential to capture and learn more complex representations and relationships within the data.

The activation functions remained unchanged, using 'relu' for both hidden layers and 'softmax' for the output layer.

The optimizer was changed to 'RMSProp'. RMSProp (Root Mean Square Propagation) is another optimization algorithm commonly used in neural networks. It adapts the learning rate for each parameter based on the average of squared gradients. This algorithm helps in mitigating the vanishing gradient problem and can lead to faster convergence in some scenarios.

These modifications in the architecture and optimizer of the ANN aimed to explore different configurations to potentially improve the model's learning capacity, representation power, and training efficiency. By increasing the number of neurons in the second hidden layer and changing the optimizer to 'RMSProp', the model's ability

Model Training			
	Raw Data	Processed Data	PCA
Training Data	0.2418	0.6483	0.6488
Validation Data	0.1067	0.6326	0 .6027

	GridSearchCV (on Processed Data)
Training Data	0.6632
Validation Data	0.6441

	Evaluation
Test Data	0.6709

Fig. 9: Artificial Neural Networks

to capture complex patterns and optimize the training process may be enhanced.

Out of all the five algorithms, we have used. It can be seen that data preprocessing helps in the performance of the algorithms.

1) The poorly performing model is OnevsRestClassifier with an F1 score of 0.485 on test data
2) The best preforming model is XGBoostClassifier with an F1 Score of 0.683 on test data

V. Future Directions

With an additional 3-6 months to work on this project, we would delve into the potential of using deep neural networks to significantly enhance our model's performance. While this approach may require considerable computational power and substantial training data, it offers promising opportunities for boosting the model's accuracy. To optimize the benefits of deep neural networks, we would collaborate with domain experts to identify the most crucial features for precise predictions. By working alongside specialists in the relevant field, we can gain a deeper understanding of the underlying processes and pinpoint the most important features for our model. This collaboration would ensure that our model is not only accurate but also practically relevant for real-world applications. Moreover, we would allocate time to fine-tune the neural network architecture and training process. This may entail experimenting with different network structures and hyperparameters to determine the most effective approach. In conclusion, adopting a deep neural network strategy would necessitate a thorough assessment of available computational resources and training data, as well as collaboration with domain experts to ensure the model's relevance and accuracy. Nonetheless, this approach holds the potential to greatly enhance the model's performance, providing valuable insights into the underlying processes.

Additionally, we have developed a user-friendly graphical user interface (GUI) using the tkinter library. Our GUI incorporates the powerful XGBoost algorithm, which has proven to be the top-performing model in our project. With this GUI, users can conveniently compare the predicted damage grade with the actual damage grade by inputting the corresponding feature values.

Furthermore, our GUI offers flexibility in data input by accepting both .csv and .txt file formats. This means that users can easily upload datasets in either of these formats, making it convenient for them to analyze and predict damage grades based on their data.

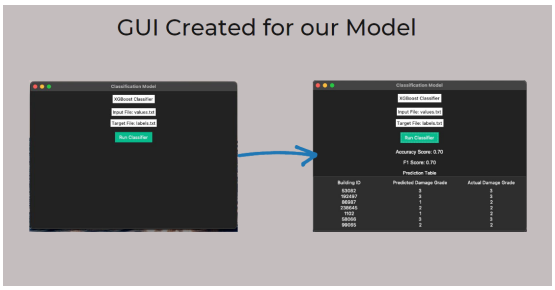


Fig. 10: GUI

VI. Conclusion

In conclusion, this project has aimed to predict the damage grade of buildings impacted by earthquakes, with the ultimate goal of providing valuable insights to urban planners, engi- neers, and disaster management agencies. Through the use of various machine learning algorithms, including Logistic Regression, Random Forests, Neural Networks and XGBoost, we have explored the potential of these models to accurately predict damage grades. We have pre-processed the data, han- dled missing values, and encoded categorical values to ensure that the models can handle the dataset effectively. The use of the micro-averaged F1 score as a metric to mea- sure the performance of our models has allowed us to evaluate their effectiveness in predicting damage grades. The results obtained can aid in developing better policies, construction guidelines, and early warning systems to minimize the loss of life and property during earthquakes. With the far-reaching consequences of earthquakes on human lives, social structures, and geographical landscapes, accurate predictions of damage grades can make a significant contribution towards effective prevention and mitigation strategies. The findings from our study have several implications:

1) Policy and Planning: Accurate prediction of building damage grades can inform urban planners, engineers, and policymakers in developing effective strategies for earthquake-resistant infrastructure and city planning. It enables them to allocate resources efficiently, prioritize high-risk areas, and implement appropriate mitigation measures.
2) Early Warning Systems: By integrating our predictive models into early warning systems, authorities can issue timely alerts to populations in earthquake-prone regions, enabling them to take necessary precautions and evacuate if required. This can significantly reduce the loss of life and minimize the impact of earthquakes.
3) Construction Guidelines: Our research findings can contribute to the development of robust construction guidelines that ensure buildings are designed to withstand seismic forces. By considering predicted damage grades, engineers can implement appropriate building materials and structural designs, enhancing the

resilience of structures in earthquake-prone regions.

References

[1] Wang, Y., Zhang, Y., Wang, Y., Shi, Y. (2019). Building damage assessment using convolu- tional neural network and post-earthquake remote sensing images. Remote Sensing, 11(18), 2148. https://doi.org/10.3390/rs11182148

[2] Huang, Y., Chen, L., Lin, J., Wei, C. (2020). Deep learning-based prediction of peak ground acceleration using seismic records. IEEE Access, 8, 134388-134397. https://doi.org/10.1109/access.2020.3016598

[3] Chong, J., Xu, H., Chang, K. C., Liu, M. (2019). Support vector machine-based prediction of earth- quake damage to buildings with building and seismic data. Journal of Performance of Constructed Facilities, 33(3), 04019016. https://doi.org/10.1061/(asce)cf.1943-5509.0001351

[4] Tymvios, N., Panayiotou, C., Ellinas, K. (2020). Random forest-based prediction of building damage probability caused by earthquakes in Cyprus. Bulletin of Earthquake Engineering, 18(3), 1213-1233. https://doi.org/10.1007/s10518-019-00709- 7