Richter's Predictor: Modeling Earthquake Damage

Contents

1 Introduction to problem

The earthquakes that occur each year cause significant economic losses. Assessing the extent of damage to buildings is essential for planning relief efforts and conducting hazard assessments. However, this task is often complex and time-consuming, making a classification system necessary. Our team will use the dataset from the Richter's Predictor: modeling Earthquake Damage competition. This is one of the largest post-disaster datasets ever collected, detailing the degree of damage suffered by over 200,000 buildings in the 2015 Nepal earthquake.

2 Objectives

Our objective is to predict the extent of damage to buildings caused by the 2015 Gorkha earthquake in Nepal. This may help government agencies in decision making and resource allocation during earthquake disasters.

3 Description of dataset

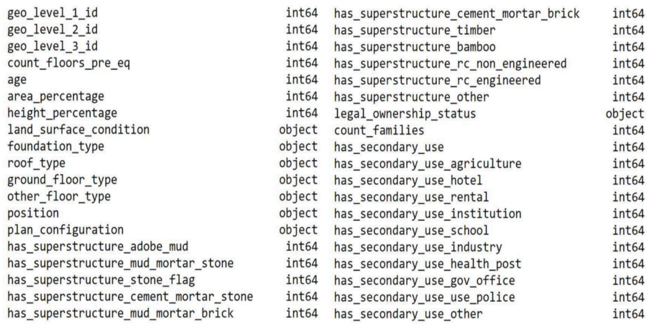
The dataset mainly contains information on building structures and their legal ownership. Each row represents a specific building in the earthquake-affected area, with 260,601 rows in total. The columns include a unique random identifier building_id and 39 variables.

3.1 Damage grade (1 target variable)

The order variable, damage_grade, which indicates a level of damage to the building that was hit by the earthquake. There are 3 grades of damage: 1 for low damage, 2 for a medium amount of damage, and 3 for almost complete destruction.

3.2 Features (38 variables)

The 38 features consist of structural information such as age, type of foundation and superstructure, as well as legal information such as ownership status, use cases, and the number of families living there. The data types are shown below, with 7 integer variables, 8 categorical variables, and 23 binary variables describing superstructure and secondary use:



```
geo_level_1_id                           int64  has_superstructure_cement_mortar_brick     int64
geo_level_2_id                           int64  has_superstructure_timber                   int64
geo_level_3_id                           int64  has_superstructure_bamboo                   int64
count_floors_pre_eq                      int64  has_superstructure_rc_non_engineered        int64
age                                      int64  has_superstructure_rc_engineered            int64
area_percentage                          int64  has_superstructure_other                    int64
height_percentage                        int64  legal_ownership_status                     object
land_surface_condition                  object  count_families                              int64
foundation_type                         object  has_secondary_use                           int64
roof_type                               object  has_secondary_use_agriculture               int64
ground_floor_type                       object  has_secondary_use_hotel                     int64
other_floor_type                        object  has_secondary_use_rental                    int64
position                                object  has_secondary_use_institution               int64
plan_configuration                      object  has_secondary_use_school                    int64
has_superstructure_adobe_mud             int64  has_secondary_use_industry                  int64
has_superstructure_mud_mortar_stone      int64  has_secondary_use_health_post               int64
has_superstructure_stone_flag            int64  has_secondary_use_gov_office                int64
has_superstructure_cement_mortar_stone   int64  has_secondary_use_use_police                int64
has_superstructure_mud_mortar_brick      int64  has_secondary_use_other                     int64
```

Figure 1: Data type

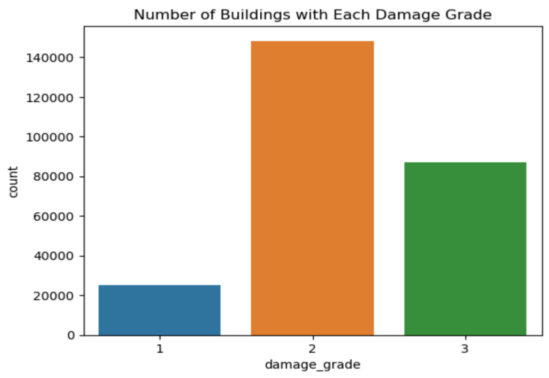
4 Methods

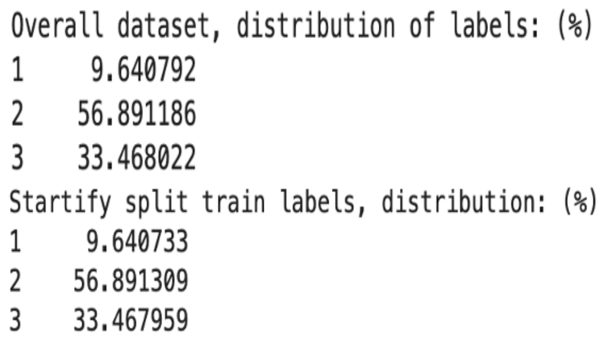
4.1 Models

To train our classifier involving ordinal variables, we have considered a range of supervised learning models: Logistic Regression, K-Nearest Neighbors (KNN), Support Vector Machine (SVM), Random Forest, XGBoost, as well as FeedForward Neural Networks given the large data size.

4.2 Train-test split

The dataset displays class imbalance, notably with class 2 being predominant. Therefore, we employed an 8:2 ratio for a stratified training-test split, resulting in training labels closely reflecting the overall dataset's label distribution.





```
Overall dataset, distribution of labels: (%)
1     9.640792
2    56.891186
3    33.468022
Startify split train labels, distribution: (%)
1     9.640733
2    56.891309
3    33.467959
```

Figure 2: Class imbalance (Left) and distribution (Right)

4.3 Performance metric

To measure the performance of our algorithms, we'll use the F1 score which balances the precision and recall of a classifier. Traditionally, the F1 score is used to evaluate performance on a binary classifier, but since we have three possible labels we will use a variant called the micro averaged F1 score.

5 Data preprocessing

5.1 Feature selection

Since this dataset has no missing values, we proceeded directly with feature engineering. Based on the correlation heatmap, it is evident that there are two groups of highly correlated features. Combining this observation with the correlation between features and our target variable, we decided to remove two features by setting a threshold of 0.7, which were 'has_secondary_use_agriculture' and 'height_percentage'.

5.2 Encoding and Standardization

We examined different types of variables. Most of the numeric variables exhibit a skewed distribution, so we considered standardizing these variables. We also performed ordinal encoding on categorical variables. These preprocessing steps were applied to all methods except tree-based models.

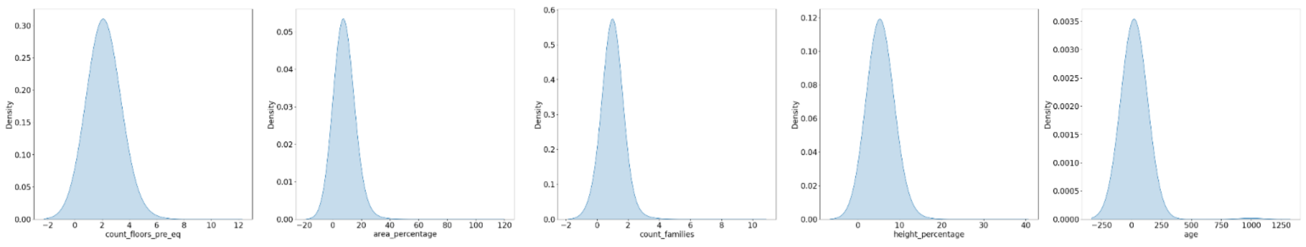
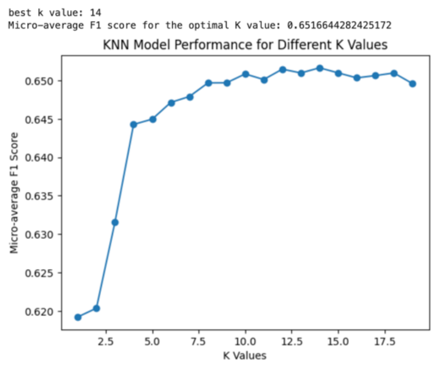


Figure 3: Distribution of numeric variables

6 Findings

6.1 KNN

The best K value is indicated as 14, which yields the highest F1 score of approximately 0.6516. The line chart shows an initial sharp increase in F1 score as K values rise from 2.5 to around 7.5, followed by a plateau, suggesting that a larger number of neighbors do not significantly improve the model's performance beyond this point.



```
best k value: 14
Micro-average F1 score for the optimal K value: 0.6516644282425172
```

Figure 4: Grid Search result for k values

The model performs best in predicting the 'Medium' category, with 79% accuracy, but has nearly equal difficulty distinguishing between 'Low' and 'High' categories, with both having a correct prediction rate of around 50%. Misclassifications are mostly between 'Low' and 'Medium' and 'Medium' and 'High' categories.

The ROC curves demonstrate the model's discrimination capacity, with the area under the curve (AUC) for class 0 being 0.87, indicating a high true positive rate against the false positive rate. Class 1 and 2 have lower AUC values of 0.71 and 0.78, respectively. The micro-average ROC curve has an AUC of 0.78, reflecting the overall performance across all classes.

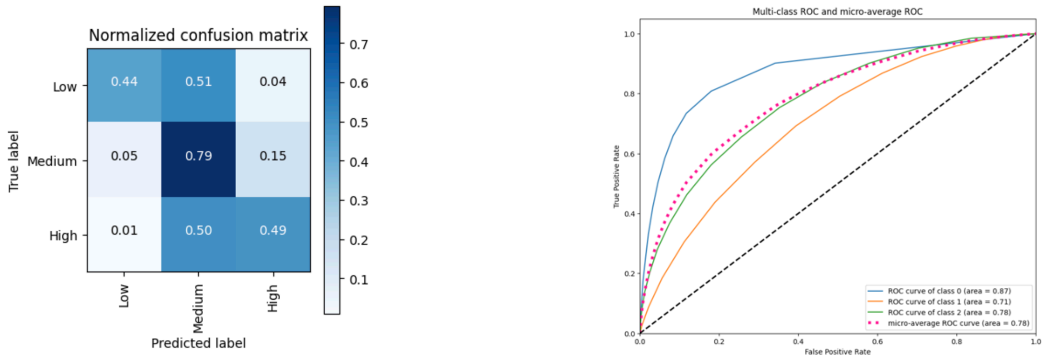


Figure 5: Confusion matrix (Left) and ROC curve (Right) for KNN

6.2 Logistic Regression

We use GridSearch CV to optimize our logistic regression model. After tuning, the best parameters obtained from the tuning process are C: 0.1 and penalty: 'l2'. The parameter C represents the inverse of regularization strength—lower values specify stronger regularization. An L2 penalty is used for regularization, which can prevent overfitting by adding a penalty for larger coefficients.

The model seems to be quite effective in predicting the 'Medium' class, as indicated by a significant number of true positives. However, it struggles with the 'Low' and 'High' classes, where it tends to incorrectly predict many instances as 'Medium'.

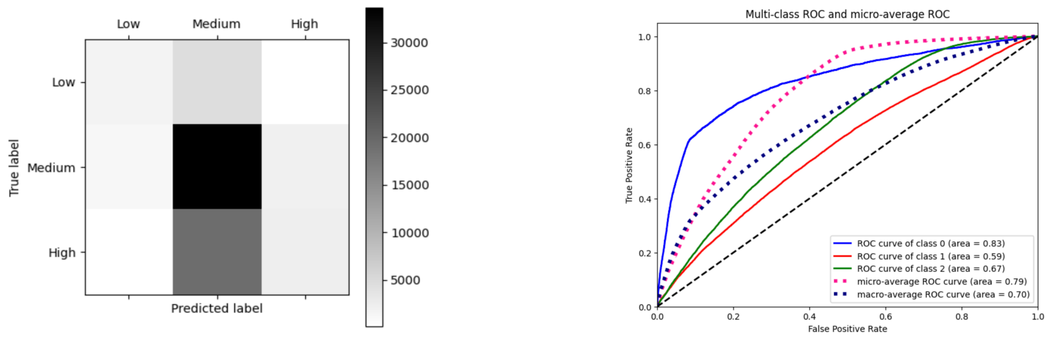


Figure 6: Confusion matrix (Left) and ROC curve (Right) for Logistic Regression

The overall F1 score of about 0.578 suggests that while the model has moderate performance, there is room for improvement, especially for the 'Low' and 'High' classes.

Class 0 has the highest AUC (0.83), signifying strong discriminatory power for this class. Class 1 has the lowest AUC (0.59), indicating that the model struggles to distinguish this class from others. The micro-average AUC (0.79) is quite good, suggesting that when the class imbalance is accounted for, the model performs well overall. The macro-average AUC (0.70) is lower than the micro-average, which may be due to the underperformance in classifying Class 1 accurately.

6.3 XGBoost

The model performed best at predicting the 'medium' class with an accuracy of 85%. The 'low' class had a 46% true positive rate, and the 'high' class had a 59% true positive rate. This matrix provides a view of the model's performance in terms of proportions, which is particularly useful for imbalanced classes.

The ROC curve illustrates the model's diagnostic ability. All three classes had similar AUC values, with Class 2 having the highest at 0.75, followed by Class 0 and Class 1 with 0.72 and 0.71 respectively. These AUC values indicate that the model has a good measure of separability and is able to distinguish between the classes reasonably well.

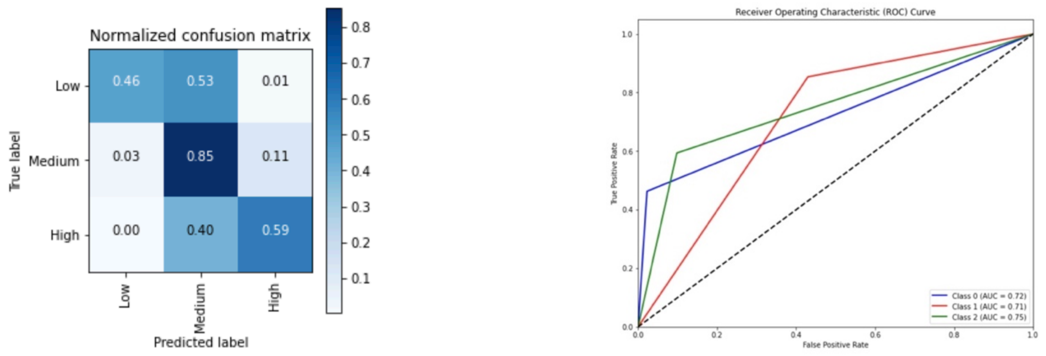


Figure 7: Confusion matrix (Left) and ROC curve (Right) for XGBoost

6.4 Support Vector Machine

The model has high accuracy for the Medium category, correctly predicting 94% of the cases. The Low category is also relatively well-predicted with 92% accuracy. The High category is correct 83% of the time but has a significant proportion of instances (16%) being misclassified as Medium.

The area under the curve (AUC) for this ROC curve is 0.80, indicating a good predictive performance. A perfect model would have a curve that goes to the top left corner, while a curve along the diagonal dashed line would indicate no better performance than random chance.

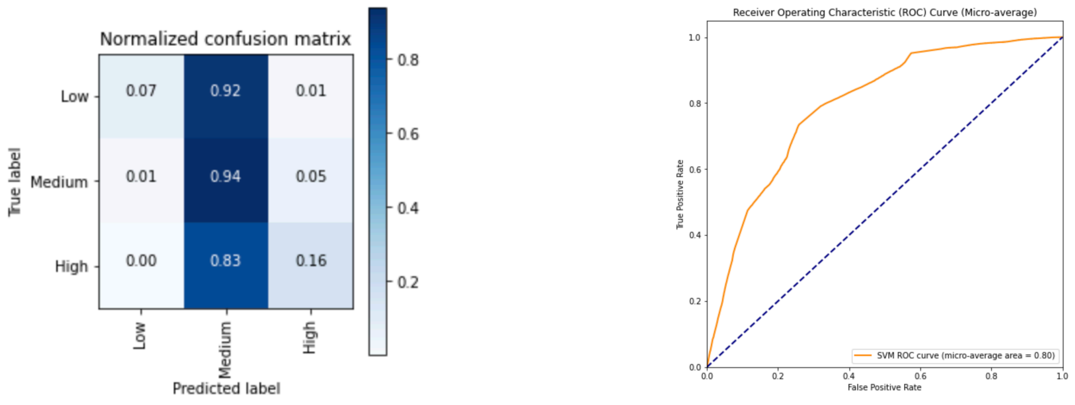


Figure 8: Confusion matrix (Left) and ROC curve (Right) for SVM

6.5 Random Forest

The result of Random Forest illustrates the impact of SMOTE and parameter tuning on our model's performance. Post-optimization, we see a significant improvement in detecting the 'Low' and 'High' categories, with recall rates rising to 0.60 and 0.68 respectively. Although the overall F1 score has slightly dipped by 0.02, the model maintains a stable accuracy of 0.74. The 'Medium' category continues to show robust performance with a recall of 0.76. This adjustment, likely through a learning curve and Bayes optimization, suggests a more balanced and nuanced model capable of better class differentiation.

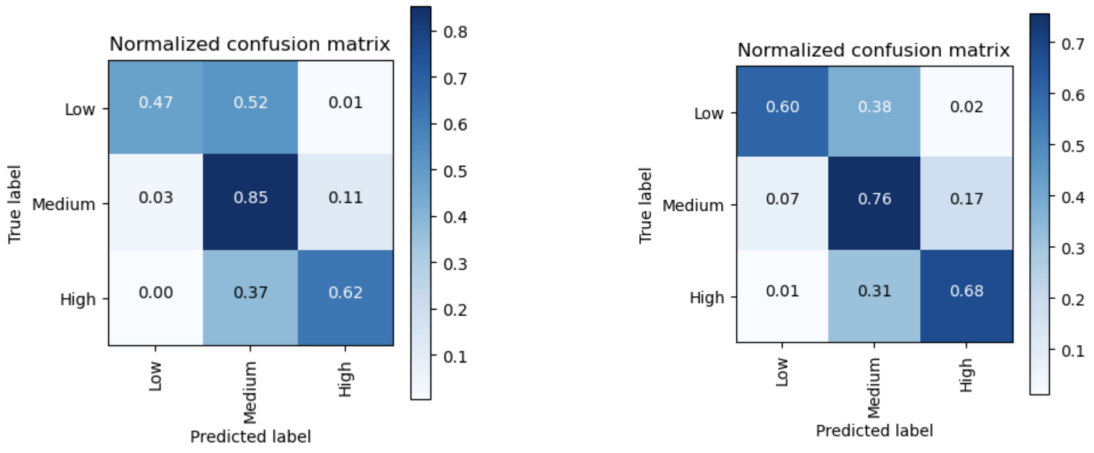
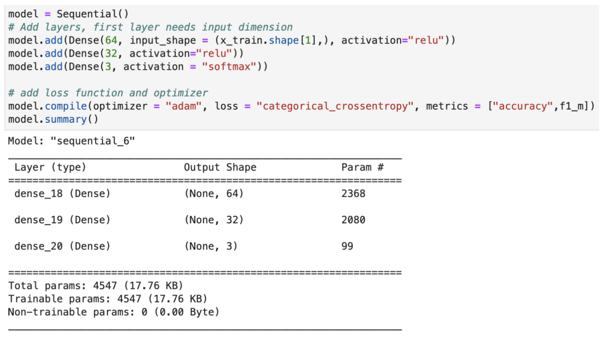


Figure 9: Confusion matrix before (Left) and after SMOTE (Right)

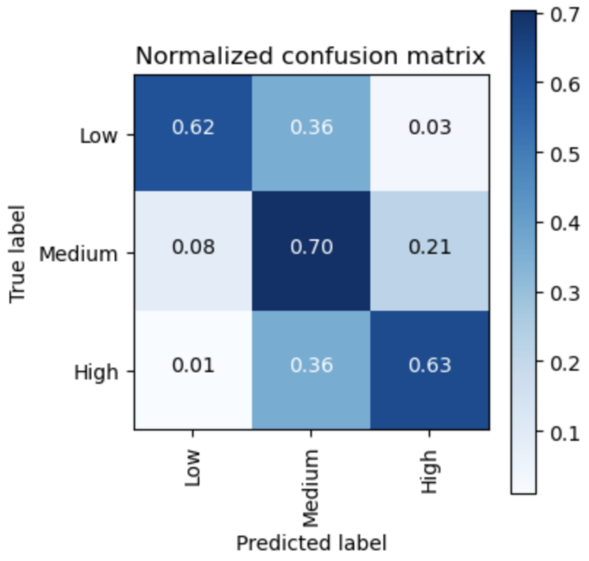
6.6 Neural networks: FFNN

Our neural network model used the Adam optimizer and categorical crossentropy for loss, with accuracy and F1 score as metrics over 30 epochs and 64 batch size. The number of input neurons is 36 and the number of output neurons is 3, because we have 36 features and 3 classes. It accurately identified 70% of Medium and 63% of High cases but confused 36% of Low as Medium. Improvement strategies include advanced feature engineering, hyperparameter tuning, network complexity enhancement, and data rebalancing to better distinguish Low from Medium categories.



```
model = Sequential()
# Add layers, first layer needs input dimension
model.add(Dense(64, input_shape = (x_train.shape[1],), activation="relu"))
model.add(Dense(32, activation="relu"))
model.add(Dense(3, activation = "softmax"))

# add loss function and optimizer
model.compile(optimizer = "adam", loss = "categorical_crossentropy", metrics = ["accuracy",f1_m])
model.summary()
```

```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_18 (Dense)            (None, 64)                2368

 dense_19 (Dense)            (None, 32)                2080

 dense_20 (Dense)            (None, 3)                 99

=================================================================
Total params: 4547 (17.76 KB)
Trainable params: 4547 (17.76 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```



6.7 Comparison between all models

1. In KNN (K-Nearest Neighbors): An F1 score of 0.65 suggests moderate performance, indicating a balance between precision and recall in classification tasks.
2. Logistic Regression: With an F1 score of 0.58, this model has room for improvement, as it may not capture the complexity inherent in predicting building damage after earthquakes.
3. SVM (Support Vector Machine): Achieving an F1 score of 0.60, the SVM indicates slightly better performance than logistic regression but still behind other models.
4. Random Forest: This model shows a high F1 score of 0.74, suggesting that it effectively captures the non-linear relationships and interactions between features.
5. XGBoost (eXtreme Gradient Boosting): With an F1 score of 0.73, XGBoost is competitive with Random Forest, indicating strong performance likely due to its robustness to overfitting and its ability to handle a variety of data structures.
6. FFNN (Feedforward Neural Network): An F1 score of 0.67 is respectable, though not the highest, suggesting that while the model can capture complex relationships in the data, there might be overfitting or a need for more nuanced network tuning.

Model	Type	Micro average F1 Score
1	KNN	0.65
2	Logistic Regression	0.58
3	SVM	0.60
4	Random Forest	0.74
5	XGBoost	0.73
6	FFNN	0.67

7 Conclusion

The ensemble methods, particularly Random Forest and XGBoost, outperformed other models in predicting the severity of building damage due to earthquakes. These models are particularly adept at handling the intricate and high-dimensional nature of such datasets. The superior scores of these models underscore their potential utility in constructing more reliable and efficient predictive tools for earthquake damage assessment. Other models, while showing potential, may require additional enhancements through advanced feature engineering, comprehensive hyperparameter tuning, or more complex model architectures to improve their predictive capabilities.

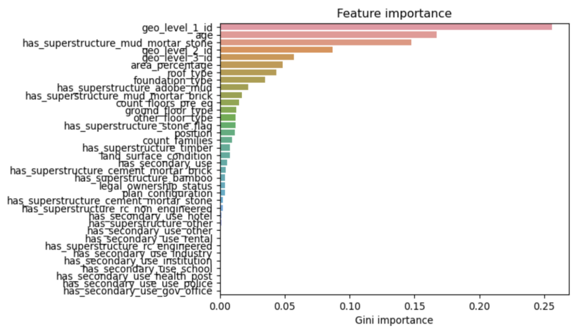
7.1 Variable importances

The left side of the graphic illustrates the feature importance based on the Gini impurity index, commonly known as Gini importance. The Gini importance measures each feature's average contribution to the purity of the node, indicating the relative importance in the Random Forest's decision trees. The geo_level_1_id emerges as the most influential feature, suggesting that this particular attribute plays a crucial role in the model's decision-making process. Other notable features include has_superstructure_mud_mortar_stone and area_percentage, which also contribute significantly to the model's performance.

On the right, the bar chart displays the feature importance according to the F score within the XGBoost framework. This metric is derived from the gain of each feature when used in constructing the boosted decision trees. Here, geo_level_2_id and geo_level_3_id show the highest importance scores, underscoring their predictive power within the XGBoost model. It is noteworthy that while there is some overlap, the order and relative importance of the features differ from those identified by the Random Forest algorithm.

The observed discrepancies between the two models' feature importance rankings could be attributed to the intrinsic differences in how these algorithms handle feature selection and weighting. Random Forest's ensemble approach, which integrates multiple decision trees to mitigate overfitting and variance, contrasts with XGBoost's sequential tree building that focuses on optimization of the loss function.

```
Text(0.5, 0, 'Gini importance')
```



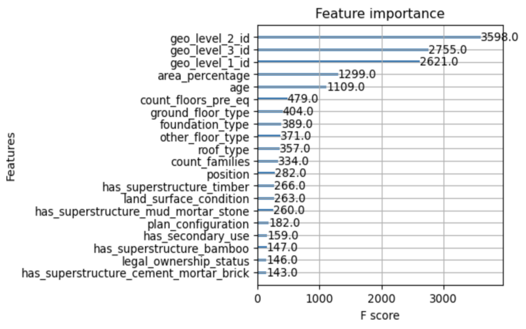


Figure 7: Feature importance of Random Forest (Left) and XGBoost (Right)

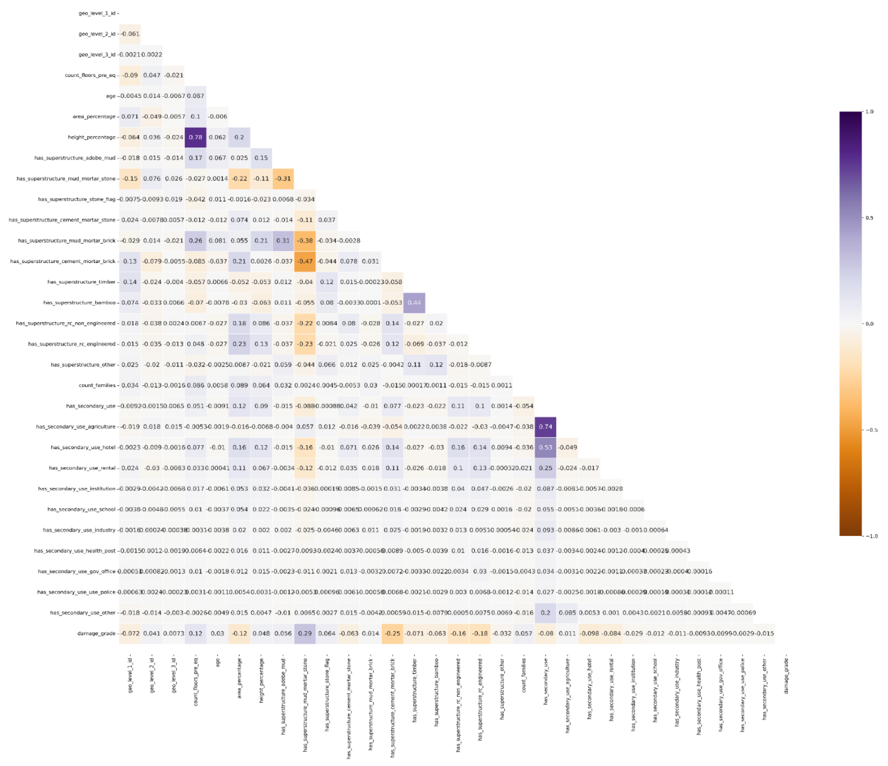
7.2 Future Improvements

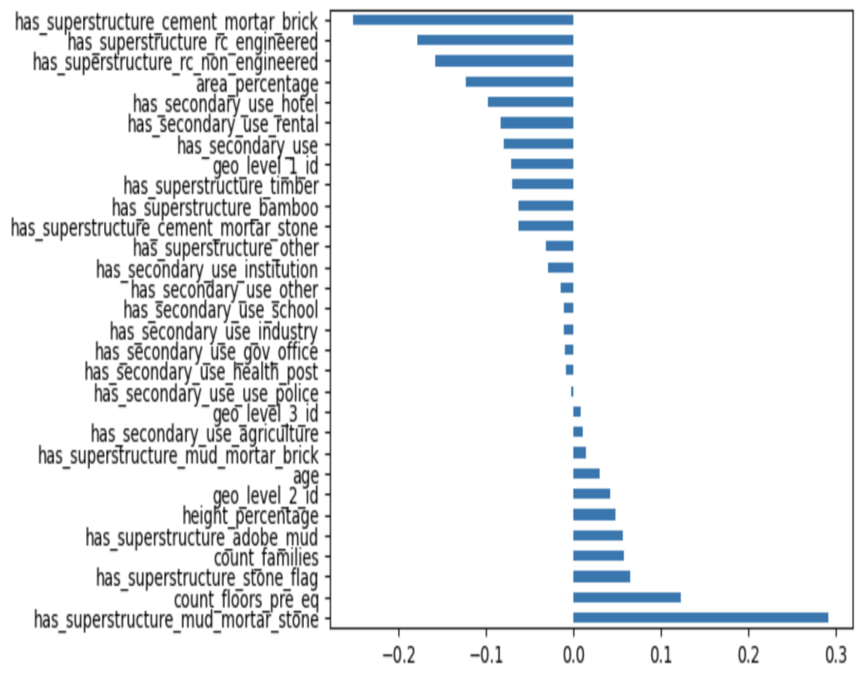
1. Due to time constraint, our exploration of the models was limited. We would have liked to do a grid search on more model parameters in the future.

2. In the future we will use more methods to deal with imbalances, such as oversampling.

3. The geo-id is crucial for predicting its damage grade, especially in relation to its proximity to an epicenter. However, the numeric format of these identifiers does not inherently correlate with the damage grade, making it challenging for a model to learn from them. We can explore this later with more coding approaches.

Appendix I – Variables correlation heatmap


geo_level_1_id
geo_level_2_id
geo_level_3_id
count_floors_pre_eq
age
area_percentage
height_percentage
has_superstructure_adobe_mud
has_superstructure_mud_mortar_stone
has_superstructure_stone_flag
has_superstructure_cement_mortar_stone
has_superstructure_mud_mortar_brick
has_superstructure_cement_mortar_brick
has_superstructure_timber
has_superstructure_bamboo
has_superstructure_rc_non_engineered
has_superstructure_rc_engineered
has_superstructure_other
count_families
has_secondary_use
has_secondary_use_agriculture
has_secondary_use_hotel
has_secondary_use_rental
has_secondary_use_institution
has_secondary_use_school
has_secondary_use_industry
has_secondary_use_health_post
has_secondary_use_gov_office
has_secondary_use_use_police
has_secondary_use_other
damage_grade


has_superstructure_cement_mortar_brick
has_superstructure_rc_engineered
has_superstructure_rc_non_engineered
area_percentage
has_secondary_use_hotel
has_secondary_use_rental
has_secondary_use
geo_level_1_id
has_superstructure_timber
has_superstructure_bamboo
has_superstructure_cement_mortar_stone
has_superstructure_other
has_secondary_use_institution
has_secondary_use_other
has_secondary_use_school
has_secondary_use_industry
has_secondary_use_gov_office
has_secondary_use_health_post
has_secondary_use_use_police
geo_level_3_id
has_secondary_use_agriculture
has_superstructure_mud_mortar_brick
age
geo_level_2_id
height_percentage
has_superstructure_adobe_mud
count_families
has_superstructure_stone_flag
count_floors_pre_eq
has_superstructure_mud_mortar_stone