# Modeling Earthquake Damage to Buildings

Xiaohan Mei
Zhixiang Ren
Wenda Zhou

## Abstract

Predicting earthquake damage grade levels is a crucial and essential research area, as it enables the anticipation of subsequent destructive consequences. This paper presents a study on modeling earthquake damage grade levels using a dataset available on the DrivenData platform, which comprises 39 features with assigned values for each row. We employ a state-of-the-art embedding technique using neural networks to extract geographical information features. Subsequently, we experimented with XGBoost, Catboost, and LightGB models, as well as an ensemble method combining these three models. The performance of these models is evaluated based on their F1 scores. The highest F1 score achieved is 75.32%, ranking 9th among all participating teams.

## 1 Introduction

The primary aim of this project is to predict the degree of damage to buildings impacted by the Nepal earthquake. To achieve this objective, our team has implemented a series of effective workflows, including data preprocessing, feature engineering, data modeling, and validation and prediction.

***Data preprocessing*** We employed a range of methods such as data cleaning, filling missing values, and handling outliers to optimize data quality. Furthermore, we used visualization and analysis to identify effective variables and data with features that can guide and direct subsequent work.

***Feature engineering*** In this project, we primarily focused on extracting geographic features and expanding data dimensionality. We utilized embedded methods to resolve geographic feature mining. These techniques can effectively improve the quality and value of the data, optimizing the predictive performance of the model.

***Data modeling*** We employed K-fold stratified sampling to prevent overfitting and underfitting. Ultimately, we selected prediction models such as Random Forest, XGBoost, and LightGBM for integration to produce more accurate and stable prediction results. Furthermore, we utilized AutoML tools to assist our team in adjusting the model parameters to improve the model's performance and efficiency, reducing the workload of manual parameter tuning. Through these tools and techniques, our team successfully completed the task of predicting the damage grade to buildings impacted by the Nepal earthquake.

## 2 Methoddology

### 2.1 Data Exploration

The dataset mainly consists of information on the buildings' structure and their legal ownership. Each row in the dataset represents a specific building in the region that was hit by Gorkha earthquake. There are 39 columns in this dataset, where the `building_id` column is a unique and random identifier. Three major issues were found in the process of the exploration and are listed as follows.

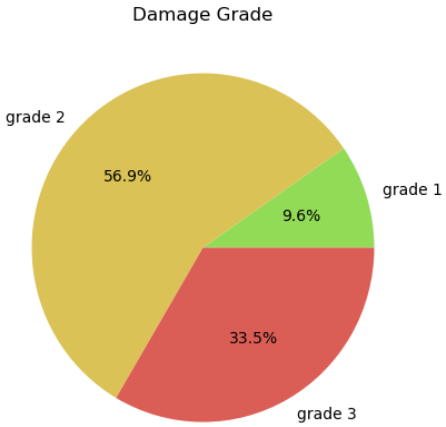


Figure 1: Class Imbalance

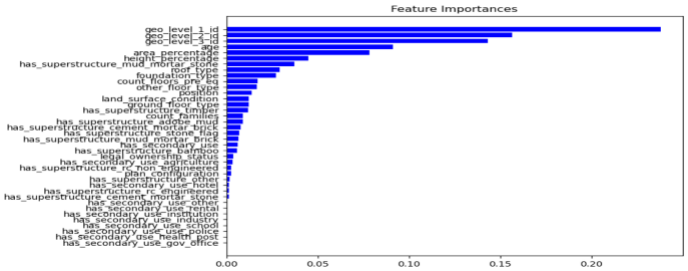


Figure 2: Feature Importance

***Class Imbalance*** This is a common issue in classification problems, as it can lead to biased models that favor the majority class and perform poorly on minority classes. In our dataset, we found that `damage_grade 2` was the majority class, while `damage_grade 1` had significantly fewer samples, as shown in *Figure 1*. A simple solution to this problem is using ensemble techniques like bagging and boosting to build multiple models and combine their predictions. These methods can be adapted to handle class imbalance. Also, this method proves to be good among our baseline models which will discussed in the below sections.

***High correlation*** High correlation between variables in a dataset indicates a strong linear relationship between them. It can be positive (both variables increase or decrease together) or negative (one variable increases as the other decreases). One example is that `ground_floor_type` and `has_superstructure_cement_mortar_brick`. However, we discovered that preprocessing for this issue was not effective, so it is advisable to leave it unaddressed.

***Feature importance*** Feature importance refers to the relative contribution of each feature to the predictive performance of a machine learning model. Understanding the importance of each feature can help in model interpretation, feature selection, and overall improvement of the model's performance. It can also provide insights into the relationships between variables and their impact on the target variable. *Figure 2* using Random Forest to generate feature importance shows that the first three rankings in our dataset are all `geo_level_id` which inspires us to make feature engineering on these features.

## 2.2 Feature Engineering

In the dataset provided, the geographic region of each building is represented by three features: `geo_level_1_id`, `geo_level_2_id`, and `geo_level_3_id`. These integer values describe the hierarchical structure of the regions, ranging from the largest (level 1) to the most specific sub-region

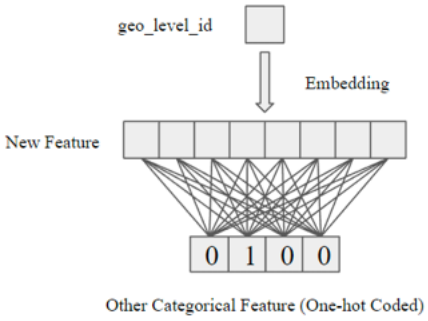


Figure 3: ANN

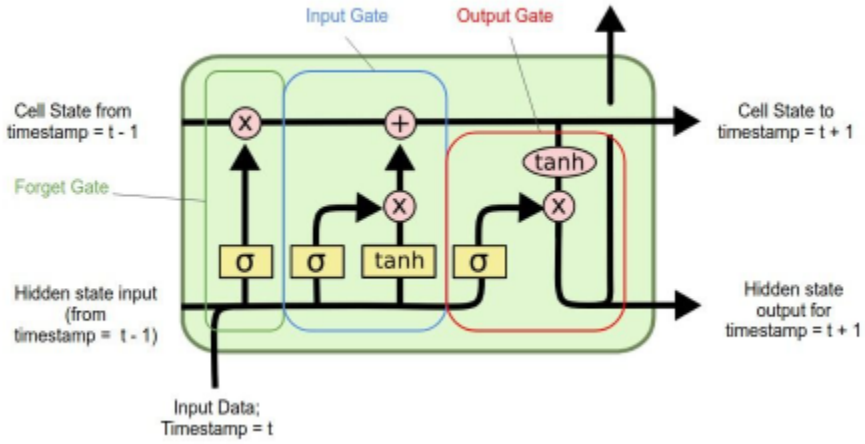


Figure 4: LSTM

(level 3). The possible values for each level are as follows: level 1 ranges from 0 to 30, level 2 from 0 to 1427, and level 3 from 0 to 12567.

Upon further examination of the data, it becomes apparent that there are distinct relationships between these three levels of geographic regions. `Geo_level_3_id` represents the smallest scale location, nested within the areas defined by `geo_level_2_id`, which in turn are contained within the larger regions represented by `geo_level_1_id`. Consequently, for any given combination of `geo_level_1_id` and `geo_level_2_id` values, there are multiple instances of the same`geo_level_3_id` value. This observation suggests a hierarchical organization of locations, with smaller sub-regions grouped under larger regions. Two machine learning models, Artificial Neural Network (ANN) and Long Short-Term Memory (LSTM), are utilized and implemented to extract valuable information from the identical data present in the dataset.

***ANN*** Autoencoders are an unsupervised learning technique in which we leverage neural networks for the task of representation learning. So we employ a Keras AutoEncoder model, using the specific location id `geo_level_3_id` as input and the larger location ids `geo_level_1_id` and `geo_level_2_id` as output. This model consists of a single hidden layer and were experimented with 8, 16, 24 neurons. Once the features are embedded, they are assigned to the training and testing data.

***LSTM*** LSTM, a type of recurrent neural network, is distinct from standard RNNs due to its feedback network. LSTMs excel in sequential data prediction, particularly in fields like machine translation. In this case, we apply a similar methodology to engineer features for the geo levels. The hierarchical structure of the geo levels suggests a sequential relationship, with level 1 being the largest region

and so forth. Consequently, we create a specialized input for the LSTM model, forming a sequence consisting of `geo_level_1_id` and `geo_level_2_id`, with the output being `geo_level_3_id`. This model were experimented with 8, 16, 24 recurrent units.

## 2.3 Model Selection and Hyperparameters Tuning

### 2.3.1 K-fold stratified sampling

K-fold stratified sampling is a cross-validation technique used in machine learning and statistical modeling to evaluate the performance and generalizability of a predictive model. It combines the benefits of K-fold cross-validation with stratification, which aims to maintain the class distribution across the different folds to ensure each fold is representative of the overall dataset. The choices of K experimented with are 5 and 10.

### 2.3.2 Loss function

Categorical Cross-Entropy Loss:

$$l = -\sum_{i=1}^{N}\sum_{k=1}^{K} y_{ik}\log(\hat{y}_{ik})$$

### 2.3.3 Baseline models

Several popular models, including Neural Networks, Naive Bayes, Random Forest, XGBoost, CatBoost, and LightGBM, have been implemented to assess their performance. The results, as presented in the following table, indicate that three gradient boosting algorithms demonstrate noticeably superior performance compared to Neural Networks, Random Forest and Naive Bayes, shown in *Table 1*. Consequently, we have decided to focus on these three algorithm types for further experimentation and analysis.

Table 1: Baseline Models

| | Neural Network | Naive Bayes | Random Forest | XGBoost | CatBoost | LightGBM |
|---|---|---|---|---|---|---|
| F1 score | 0.6816 | 0.5615 | 0.7420 | 0.7456 | 0.7460 | 0.7517 |

### 2.3.4 Model Selection

***XGBoost*** short for Extreme Gradient Boosting, builds on the foundations of gradient boosting, incorporating regularization terms to control model complexity and prevent overfitting. One of the key features of XGBoost is its ability to handle sparse data, making it particularly well-suited for datasets with missing values or high-dimensional feature spaces.

Hyperparameters:
`learning_rate`: 0.05, 0.1, 0.12, 0.14, 0.18
`max_depth`: -1, 2, 4, 6, 8
`n_estimators`: 100, 200, 300, 500, 1000, 1500, 2000

***CatBoost*** is a cutting-edge gradient boosting algorithm specifically designed to handle categorical features without the need for extensive preprocessing. Additionally, CatBoost employs an ordered boosting technique to combat overfitting, enabling it to achieve excellent performance even in cases where the dataset has a high degree of noise.

Hyperparameters:
`learning_rate`: 0.08, 0.1, 0.12, 0.14, 0.18
`iterations`: 500, 800, 1000, 1500, 2000
`depth`: 2, 4, 6, 8

***LightGBM***, or Light Gradient Boosting Machine, introduces gradient-based one-side sampling (GOSS), which selectively samples instances for gradient updates based on their gradients, allowing the algorithm to focus on more informative instances while maintaining accuracy. Another notable

feature of LightGBM is exclusive feature bundling (EFB), which groups mutually exclusive features together, reducing the dimensionality of the feature space and lowering memory usage.

Hyperparameters:
`learning_rate`: 0.08, 0.1, 0.12, 0.14, 0.18
`max_depth`: -1, 2, 4, 6, 8

### 2.4 Ensemble

We further explore and experiment with ensemble method by using the results of Random Forest, XGBoost, LightGBM to obtain better predictive performance.The ensemble of these three models represents the best combination we obtained during the experiment. Each model was given the likelihood of the damage grade and then sum up likelihood with weights to pick the most likely one.

## 3 Experimental Results

### 3.1 Results prior to fine-tuning

Table 2: F1 score of the models on validaiton set

| K-fold | Embedding method | Random Forest | XGBoost | Catboost | LightGBM | Ensemble method |
|---|---|---|---|---|---|---|
| 3 | ANN | 0.7926 | 0.8078 | 0.8072 | 0.8061 | 0.8127 |
| 3 | LSTM | 0.7606 | 0.7671 | 0.7736 | 0.7774 | 0.7695 |
| 5 | ANN | 0.7952 | 0.8111 | 0.8326 | 0.8344 | 0.8454 |
| 5 | LSTM | 0.7656 | 0.7786 | 0.7901 | 0.7909 | 0.7901 |
| 10 | ANN | 0.8008 | 0.8126 | 0.8240 | 0.8264 | 0.8293 |
| 10 | LSTM | 0.8023 | 0.8094 | 0.8098 | 0.8120 | 0.8115 |

*Table 3* presents the optimal F1-score achieved on our validation set for each model, considering various K-fold selections and embedding methods prior to fine-tuning the hyperparameters. Although we experimented with different numbers of neurons for embedding methods, we have not included these results in the table, as all the reported F1-scores utilized embedding methods with 16 neurons.

In the context of embedding method model selection, we observed that ANN consistently outperformed LSTM, leading us to choose ANN as the embedding method for feature engineering in our submission.

With respect to selecting an appropriate K (number of folds), we found that while increasing K could improve the F1-score, a high F1-score did not guarantee similar performance on the test set. In fact, overfitting became a potential issue. Consequently, we selected a suitable number of folds to ensure more effective model training and minimize overfitting.

### 3.2 Results with fine-tuning

Table 3: F1-score of fine-tuned ensemble method and LightGBM

| | Models | F1 score |
|---|---|---|
| / | LightGBM | 0.7510 |
| Fine-tuned | LightGBM | 0.7528 |
| / | Ensemble Mehtod | 0.7512 |
| Fine-tuned | Ensemble Method | 0.7532 |

Upon fine-tuning the models, we determined that the Ensemble Method emerged as the top-performing model, achieving an F1-score of 0.7532. This model employed the following hyperparameters: a `learning_rate` of 0.05 for the Random Forest, 0.1 for both XGBoost and LightGBM; a `max_depth` of 10 for XGBoost and -1 for LightGBM; and a total of 2,000 estimators `n_estimators`. This result not only reflects the model's performance but also accounts for factors such as interpretability, training and prediction efficiency, model stability, and scalability.

# 4 Discussion

We explored various methods throughout each stage of the project. During the data preprocessing phase, we employed multiple visualization techniques to uncover insights. However, we discovered that the data distribution across dimensions did not follow any apparent patterns, and there were certain flaws such as imbalanced degrees of damage in the samples. These issues posed significant challenges during the data preprocessing process. We also attempted to use clustering analysis on unlabeled samples to uncover patterns, but the results were not dependable.

In the feature engineering stage, we experimented with various methods to extract features from geographical and other data. We designed a deep neural network using the embedded method to extract features that combined geographical and other data in higher dimensions. Unfortunately, this approach did not improve our accuracy. We also tried recoding geographical data to restore and reconstruct its context. However, this method actually eliminated the features in the geographical data and had no significant effect. Similarly, when we attempted to convert the actual IDs of geographical data into frequency and occurrence, the results lost the positional and distance information, causing a decrease in the model's accuracy.

During the data modeling stage, we experimented with various methods. We attempted to use the naive Bayes and neural network methods to predict the degree of damage, but no matter how we tuned them, the accuracy remained below 0.7. We employed the Adaboost method for prediction, but its results were not as good as LightGBM. Although we tried using SVM for classification, it was inefficient and performed poorly when processing large sample data. We attempted logistic regression for classification, but only the constant term was significant, and the model's R-squared was very small. Even using polynomials for nonlinear regression did not yield satisfactory results. Therefore, we were constrained in our selection of models and methods. We tried using random search and grid search to fine-tune the model parameters, but they did not help us effectively complete the task.

During the validation and prediction stage, we encountered imbalanced data, so we used stratified sampling to avoid erroneous predictions due to data imbalance. We also attempted to find the optimal algorithm combination through weighting methods.

Throughout the project, we also acquired general knowledge about evaluating earthquake hazards and the degree of damage. We consulted relevant research papers, but most of the methods were not applicable to our dataset. If the dataset had included latitude and longitude data, we could have used ArcGIS to assist with analysis. Unfortunately, the original dataset did not contain such data, and we were unable to use external data.

# 5 How to run the code

1. Before running the code, please ensure that the raw data file, `submission_format`.csv file, and the code file are placed in the same folder.
2. To train and save the model, create a new folder named "models" in the same directory.
3. Before running the code, make sure that the necessary libraries for the program are installed, e.g., numpy, pandas, scikit-learn, lightgbm, etc.
4. Run all code cells. (Cells below "Catboost" do not need to be run.)