# Application and optimization of machine learning models to predict building damage level using 2015 Nepal earthquake statistics

**Aaron Dardik**
Department of Computer Science
University of Southern California
Los Angles, CA 90086
dardik@usc.edu

**Jonathan R. Kasprisin**
Department of Computer Science
University of Southern California
Los Angles, CA 90086
kasprisi@usc.edu

**Shriya Nagrath**
Department of Computer Science
University of Southern California
Los Angles, CA 90086
nagrath@usc.edu

## Abstract

In this paper, we investigate multiple supervised-learning models and measure their performance in the task of predicting earthquake damage grade based on data from Nepal's Gorkha earthquake in April 2015. We began with exploratory data analysis with the aim of understanding the features in our dataset and their relative importance in predicting damage grades to buildings in Nepal. After cleaning, preparing and encoding our data we tested several models. These ranged from neural networks to boosting models like LightGBM, XGBoost, and CatBoost to a combination of these models adapted into a metaclassifier. Our most successful model was CatBoost, with a micro f1 score of 0.7511. Our CatBoost model, when trained on DrivenData's training dataset is in the top decile of competitors in Richter's Predictor: Modeling Earthquake Damage (as of 4/21/2023).

## 1 Introduction

The ability to predict where the most damage from an earthquake will occur is key to ensuring the efficient allocation of resources in the wake of natural disasters. Richter's Predictor: Modeling Earthquake Damage is a challenge on DrivenData.org based on the April 2015 Gorkha earthquake that struck central Nepal, killing 8,964 people. The goal of this competition is to accurately predict the damage done to buildings across Nepal based on tabular data collected after the disaster. There are many possible approaches to creating a learning model from a tabluar dataset. Data cleaning, feature preparation, model selection, and hyperparameter tuning all affect a model's efficacy. In this paper, we will analyze various models and techniques to identify best practices to achieve a precise multi-class classification on this type of tabular data.

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

## 2 Exploratory data analysis and feature preparation

### 2.1 The data set

The data set provided by the DrivenData official website [1] was curated through a collaborative effort between Kathmandu Living Labs and the Central Bureau of Statistics. It comprises an extensive range of socio-economic, demographic, architectural, topographical, and geographical information collected following the 2015 earthquake. This data set encompassed 38 features and an aggregate of 260,601 input vectors with ordinal output labels of damage grade. The test dataset consisted of 86,868 input vectors which are used to evaluate model performance. We conducted exploratory data analysis to identify processing based on the model type and possible areas for improving performance with feature engineering.

### 2.2 Exploratory data analysis(EDA)

From a comprehensive overview of the EDA in Appendix B, we found the dataset was characterized by a class distribution of approximately 33.4% high damage, 9.6% low damage, and 56.8% medium damage. This indicated an imbalanced dataset, potentially limiting any selected model's ability to generalize. The dataset required no cleaning since there were no missing entries in the training or competition dataset. The dataset consisted of five numerical, three high cardinality categorical features, and thirty low cardinality categorical features. Preliminary analysis of the input features and the labels indicated that none of the features displayed a strong correlation, as reflected in Figure 1 . Due to low correlation, we used model feature importance and researched encoding methodologies to capture latent characteristics. Additionally, we observed that all the numerical features had heavily skewed distributions, with the feature 'age' having the only outlier. We tested if feature engineering schemes to correct for all of these observations resulted in improved model scores.

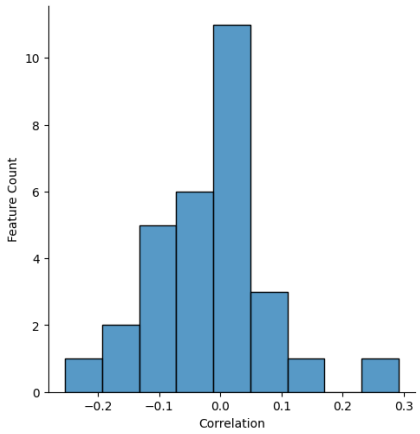


Figure 1: Correlation histogram illustrates how many features have each value of Pearson's coefficient

### 2.3 Feature engineering

In our research, we tested different data pre-processing schemes to evaluate their impact on model performance. The following transformations were tested:

**Numerical features** We applied ScikitLearn's (sklearn) StandardScaler to see if model performance improved with scaling. All numerical feature distributions were found to be heavily skewed, so we also tried applying log transformation before StandardScaler.

**Low cardinality categorical features** For all models we tried sklearn's OneHotEncoder and sklearn's LabelEncoder, and if available, we tried native handling of categorical variables.

**High cardinality categorical features** The Geo_id features were all high cardinality. For these features, we tried native categorical handling and Guillermo Navas-Palencia OptBinning. Upon

further review of the data and prior work [2], we tried capturing the latent relationship between the features and labels with sklearn's TargetEncoder, MEstimatorEncoder, JamesStienEncoder, and CatboostEncoder. We assigned each geographic location feature a unique value and used auto-encoding with a single hidden layer with sixteen neurons[3], binary cross-entropy loss function, and adam optimizer. We also used Long Short-Term Memory architecture to learn geo_level_3, given geo_level_1 and geo_level_2[4].

**Class balancing** We tested native model minority class sampling and Synthetic Minority Over-sampling Technique (SMOTE) with RandomUnderSampler and K-Nearest neighbor.

## 3 Methods and findings

For all experiments, we ran single trials to understand changes rather than repeat trials to test robustness so scores do not have an error margin. Detailed results by experiment are available in Appendix C.

### 3.1 Baseline models methodology and findings

Our initial methodology was to establish a baseline performance using several basic models. From the sklearn library, we chose the Random Forest, AdaBoost, and Logistic Regression classifiers. This compared decision tree bagging, decision tree boosting, and a one-vs-rest (OvR) scheme with cross-entropy loss methods. We used the same preprocessing for all models with StandardScalar for numerical features, OneHotEncoder for Categorical Features, and the Geo_ID features treated numerically and GridSearch for optimization. The training data was split 25% for testing with the total F1 Score as the evaluation metric using no cross-validation. After later experiments with boosted models, we returned to random forest using MEstimatorEncoder from the for the Geo_ID features, StandardScaler for numerical, and getDummies for the categorical features. See Table1 for model parameters selected for optimization using Optuna. We ran Optuna optimization for 20 trials.

From our initial experiments, we determined that methodologies based on decision tree weak learners returned the best results but overfitted significantly in the case of directly applying Random Forest. Due to the overfitting seen in the Random Forest Classifier, we focused on boosting models. When we returned to Random Forest using MEstimatorEncoder, we found the results close to the performance of boosting models with the best cross-validated F1 Micro score of 0.751 and competition F1 Micro Score of 0.735.

Table 1: Model Parameters

| Model | Optimal Parameters |
|---|---|
| LightGBM | lambda_l1=0.00006, lambda_l2=4.9, num_leaves=99, bagging_freq=3, min_child_samples=40, max_depth=9, feature_fraction=0.37 |
| XGBoost | booster=dart, lambda=0.0009, alpha=0.04, subsample=0.48, col-sample_bytree=0.54, max_depth=7, min_child_weight=3, eta=0.10, gamma=1.2, rate_drop4.69, skip_drop=0.29 |
| Random Forest | n_estimators=61, max_depth=12, max_features=0.34 |
| CatBoost | iterations=3000, max_depth=9, eval_metric='TotalF1', l2_leaf_reg=3, learning_rate=0.05, loss_function='MultiClass', task_type="CPU" |

### 3.2 Neural Networks methodology and findings

We experimented with using a neural network to predict damage grade. We applied StandardScaler for numerical features and OneHotEncoder for categorical features. The processes of feature selection and hyperparameter optimization can be thought of as repeatedly performing two grid searches, searching for the greedy optimum along the trajectory implied by the previous search function's greedy optimum. We consider the first set of hyperparameters to be the number of non-output layers in the neural network, and which features to include in the model. Finding the greedy optimum of this first search function, $s_1$ on $\mathbb{Z}_+ \times \{0,1\}^{|\{\text{features}\}|}$ is accomplished via grid search. The second set consists of the size of each layer of the neural network and the possible activation functions between

layers. To narrow down the search space across possible activation functions we fix the function to the output layer to be the softmax function and restrict the other activation functions to be from the set {sigmoid, ReLU, tanh} which is isomorphic to $\mathbb{Z}_3$ and will hereafter be denoted as such. The search function $s_2$, on $\mathbb{Z}_+^n \times \mathbb{Z}_3^n$ (where $n$ is the projection of $s_1$ onto its first coordinate) is also optimized via grid search.

Repeatedly alternating between finding greedy optima for the two search functions results in a network with 4 non-output layers, including all features except those that begin with the expression 'has_secondary_use', hidden layer sizes of 90, 90, 45 and activation functions (including between input and hidden layers as well as hidden and output layers) of ReLU, tanh, ReLU, ReLU, softmax. This resulted in a neural network with a test f1 score of 0.68. Further improvements in training f1 resulted in declines in test f1, suggesting the network was overfitting to the training data. In light of the overfitting issue, we decided to devote our time to more promising models.

### 3.3 Boosted decision tree methodology and findings

#### 3.3.1 LightGBM and XGBoost

Based on our findings from basic models, we conducted a series of experiments using the more advanced boosted decision tree models CatBoost, LightGBM, and XGBoost. For all experiments, we used StandardScaler for numerical features. For LightGBM, we ran a series of experiments to test boosted model performance under different conditions. Tests included dropping low-importance features, changing categorical feature encoding, changing target encoding for Geo_ID features, and weighted sampling based on potential class imbalance. For XGBoost, we ran two experiments, the first enabling the model to do encoding internally and the second using the best encoding scheme from LightGBM. XGBoost was always run with GPU-Hist. All LightGBM and XGBoost runs were tuned with Optuna running 100 trials with tuned parameters shown in Table1. Scoring was done using five-fold cross-validation of the F1 Micro score and compared to F1 Micro Score with the model predicting competition data.

From these experiments, we found that for the given dataset, the most impactful change was using a target encoding methodology for the Geo_ID features. Of the target encoding schemes experimented with MEstimatorEncoder performed the best for LightGBM and XGBoost. Dropping unimportant features such as 'has_secondary' using Recursive Feature Eliminator had a very minor negative impact. Adjusting for skewed numerical features had no noticeable effect. Addressing the class imbalance hurt XGBoost and had no noticeable impact on LightGBM. Tuning model parameters with Optuna had a minor positive impact.

#### 3.3.2 CatBoost

The CatBoost[5] algorithm is available as a powerful open-source library that utilizes gradient boosting on decision trees, and performs well on categorical data. We discovered that CatBoost performed decently on the dataset using default parameters resulting in an F1 score of 71.22%. We focused on a range of parameters to enhance performance, including categorical features (cat_features), loss function (loss_function), processor type (task_type), and evaluation metric (eval_metric). We manually modified these parameters and employed automation tools like autoML, optuna[6], and GridSearch.Feature reduction and non-native encoding schemes resulted in worse performance. We achieved the best model and overall performance using CatBoost native encoding with parameters optimized using GridSearch running on CPU for a score of 0.7511.

We also observed that CatBoost performs significantly better when trained on CPU rather than GPU. While CatBoost's documentation did not address this issue, it is possible the variation was due to the difference in default parameters associated with both processing units [5].

### 3.4 Meta-classifier methodology and findings

Since several models had similar F1 Micro performance, we tried using a stack and ensemble metaclassifier, combining the best models: LightGBM, RandomForest, CatBoost, and XGBoost. For the ensemble, we used the sklearn VotingClassifer with soft voting and weighing CatBoost since it had the highest individual performance. The ensemble was evaluated using cross-fold validation and then trained on 100% of the training data before predicting the competition data labels. We used

sklearn StackingClassifer with LogisiticRegression as the final estimator and five folds for the stack model. For all metaclassifiers, we used StandardScaler for numerical features, MEstimateEncoder for target Encoding the Geo_ID features, and LabelEncoder for the categorical features. The best parameters were selected from individual Optuna runs for each model. CatBoost and XGBoost used GPU.

We found that the stack and ensemble performed the same on the training data, but the stack metaclassifier performed slightly better on the competition data. Both metaclassifiers performed worse than the best individual classifier.

### 3.5 Running experiment code

Results can be reproduced with notebooks in the Appendices ran in GoogleColab where the dataset will need to be retrieved from DataDriven [1] and pathname updated. Best results were achieved using Appendix A, "FINAL_submission_Catboost". For this file all prior experiments and sub-optimal modifications have been commented out. Kindly do not change the default run time type.

## 4 Results and discussion

As summarized in Figure 4, decision tree models achieved the best overall performance on the testing set. As prior research suggests that this is due to the inductive biases in decision tree-based models [7]. Our research did not further examine the mechanism causing decision trees to perform better so this a potential area for future research. Our best performance came from an individual classifier, CatBoost, with optimized hyperparameters rather than any of the stack or ensemble methods. This could be caused by the four base classifiers having similar output since they all had the same input vectors. Future research should examine if using k-fold split of the training set for each base classifier in the stack or ensemble improved performance. Another point of interest is CatBoost was the best-performing model rather than a different boosting model. This could be due to the fact that the majority of our dataset features were categorical, and CatBoost was designed for categorical features, but further research is required [5]. An additional limitation of our work is that experiments were run with a single trial, so there could be variance in performance by model.

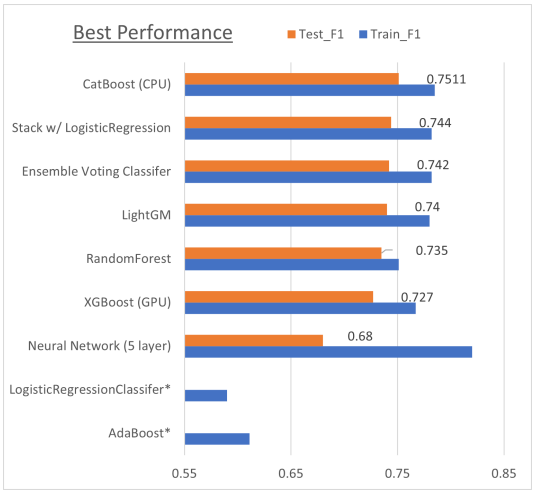


Figure 2: Summary of the best performance for each model tried. Displayed score if from competition data. *no competition score

## 5 Conclusion

Assessing damage grade to Nepal's infrastructure is a challenging task due to low feature-label correlation, high levels of overfitting to the training dataset, and latent geographical data in Geo_level_id features. We found that the neural network's weakness on tabular datasets made performance worse

than decision tree-based methods. As decision tree-based models proved promising, we sought to further increase performance through feature encoding, data pre-processing, feature reduction, minority class sampling, and tuning model hyperparameters. Appropriate feature encoding and optimizing model parameters resulted in increased performance. Our research testing ensemble and stacking methods never outperformed the strongest individual in the ensemble. In our tests, the best performance was from an optimized Catboost running on CPU to achieve an F1 Micro score of 0.7511. Future attempts to further improve performance could target improvements in individual boosting models by refining the auto-encoding of features. Another future avenue for improving performance would be to reduce overfitting by creating an ensemble from folds of the dataset rather than using the full data set for each model.

## References


[1] DrivenData. Richter's predictor: Modeling earthquake damage -data download. https://www.drivendata.org/competitions/57/nepal-earthquake/data/. Accessed April 2, 2023.

[2] P. Mora. Richter's predictor- data challenge from drivendata. *Econometrics & Data Science*, June 2020. Accessed April 10, 2023.

[3] Kutsal Ozkurt. Richter's eye. https://github.com/Goodsea/Richter-s-Eye/blob/master/main.ipynb. Accessed April 21, 2023.

[4] R. Prashnani. Geograhical feature encoding: Lstm. https://medium.com/swlh/predicting-damage-to-building-due-to-earthquake-using-data-science-e85a62adc0c0. Accessed April 19, 2023.

[5] A.V. Dorogush, V Ershov, and A Gulin. Catboost: gradient boosting with categorical features support. *arXiv preprint arXiv:1810.11363*, 2018.

[6] Optuna. Optuna-examples/catboost_pruning.py at main · optuna/optuna-examples. https://github.com/optuna/optuna-examples/blob/main/catboost/catboost_pruning.py. Accessed April 19, 2023.

[7] R Shwartz-Ziv and A. Armon. Tabular data: Deep learning is not all you need, 2021.

[8] P. Bull, I. Slavitt, and G. Lipstein. Harnessing the power of the crowd to increase capacity for data science in the social sector. *arXiv preprint arXiv:1606.07781*, June 2016.

[9] M. Clarke. How to tune a lightgbmclassifier model with optuna. *Practical Data Science*, January 2023. Accessed April 19, 2023.

[10] C. Fitzpatrick. Richter's predictor - benchmark. https://drivendata.co/blog/richters-predictor-benchmark/. Accessed April 2, 2023.

[11] P. Banerjee. Catboost classifier in python. https://www.kaggle.com/code/prashant111/catboost-classifier-in-python. Accessed April 5, 2023.

[12] R. Paul. Tutorial lightgbm + xgboost + catboost (top 11%). https://www.kaggle.com/code/paulrohan2020/tutorial-lightgbm-xgboost-catboost-top-11/notebook, April 2022. Retrieved April 21, 2023.

[13] L. Shukla. Designing your neural networks. https://towardsdatascience.com/designing-your-neural-networks-a5e4617027ed, September 2019. Retrieved April 10, 2023.


## A Final submission code

## B EDA notebook

## C Experiment parameters and results

## Checklist

1. For all authors...

(a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]

(b) Did you describe the limitations of your work? See section 4

(c) Did you discuss any potential negative societal impacts of your work? [N/A]

(d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]

2. If you are including theoretical results...

(a) Did you state the full set of assumptions of all theoretical results? [N/A]

(b) Did you include complete proofs of all theoretical results? [N/A]

3. If you ran experiments...

(a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] Code to reproduce results was provided in Appendix A and instructions in Section 3.5

(b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See Section 3 and Appendix C

(c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [N/A] Address in limitations of our work Section 3 and Section 4

(d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Section 3 and Appendix C

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...

(a) If your work uses existing assets, did you cite the creators? [Yes] See references for Optuna, CatBoost, LightGBM and XGBoost

(b) Did you mention the license of the assets? [Yes]

(c) Did you include any new assets either in the supplemental material or as a URL? [Yes] See Appendix A

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A] We used publicly available datasets

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A] We used publicly available datasets

5. If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]