Team 11: Using Weighted Ensemble to Predict Earthquake Damage

Brian Hyeongseok Kim
brian.hs.kim@usc.edu

Shivam Kotak
skotak@usc.edu

Ujjwal Pasupulety
upasupul@usc.edu

Abstract

The assessment of building damage caused by disastrous events such as earthquakes is time-consuming but needs to be done accurately. This project aims to create a machine learning model that will efficiently solve the problem of structural damage level prediction on the devastating earthquake that impacted Nepal in 2015. This paper explores the factors that influence the prediction accuracy through data preprocessing, feature selection, hyperparameter tuning, and model ensembles. We evaluate the efficacy of our trained models through an open DrivenData competition [1]. Our final result, evaluated with micro F1 score, was **0.7527**.

1 Introduction

In April 2015, Nepal was struck with a devastating earthquake that damaged thousands of buildings and resulted in even more fatalities and injuries. As a response, Kathmandu Living Labs and the Central Bureau of Statistics generated a large dataset that includes information on earthquake impacts, household conditions, and socio-economic-demographic statistics. One of the largest of its kind, it has quickly become a standard dataset on which scholars and organizations evaluate their earthquake analysis methods. More specifically, there exist many papers that train different machine learning models on this dataset to quickly classify post-earthquake damage to buildings.

Our paper describes one such machine learning solution that we devised, as part of an ongoing DrivenData competition called Richter's Predictor: Modeling Earthquake Damage. Using their provided training and testing datasets, we employ five different machine learning models, Random Forest (RF), Extra Trees (ET), XGBoost, LightGBM, and CatBoost, and evaluate their prediction accuracy on the level of building damage.

2 Related Work

There are many studies that employ machine learning to predict the level of building damage using data of the 2015 Nepal earthquake. For example, several works [4, 1, 3] conclude that Random Forest seems to output high prediction accuracy (68%, 74%, and 77% respectively). A synthesized study of multiple earthquakes around the world concludes that for the 2015 Nepal earthquake, both Random Forest and Extra Trees report high prediction accuracy at around 72%, with Extra Trees having a slightly higher accuracy average when all the studied earthquakes are considered [5]. Other works [6, 2, 7] state that using XGBoost, which is a decision tree-based algorithm that uses gradient boosting, seems to output a higher prediction accuracy than Random Forest (58%, 73%, and 74% respectively). With these works in mind, we decide to initially explore these three machine learning models, as described in Section 4.

[1] https://www.drivendata.org/competitions/57/nepal-earthquake/

3 Dataset

We obtained the dataset through the DrivenData competition. It consists of three CSV files, one for train values, train labels, and test values. The dataset contains information about 347,469 buildings that were impacted by the 2015 Nepal earthquake. The training and testing sets have been split as 75-25, resulting in 260,601 and 86,868 buildings, respectively. In all of the files, each row represents a building, identified by a unique index *building_id*.

The *train_values.csv* file provides information about 260601 buildings through 38 features. These features provide structural information (e.g. number of floors, building age, and foundation type) and other information (e.g. ownership status, building use, and number of residential families). There are 260,601 rows for this training dataset. One thing to note is that for categorical features, some manipulation needs to be done so the models can read and understand the values. A popular method for this is one-hot encoding.

In *train_labels.csv*, there are 2 columns, *building_id* and *damage_grade*. The latter is essentially the label we want our model to predict, where a 1 represents low damage, 2 medium, and 3 almost complete destruction. Because this is for the training dataset, there are 260,601 rows as well.

The *test_values.csv* file is similar to *train_values.csv* file, but it contains information about 86,868 buildings that were not part of the training set. This testing set is used by our trained model to make predictions. There are no labels made available for this testing set, so we have to submit our predictions to the competition. Since DrivenData uses micro F1 score as the evaluation metric, we use the same to discuss our evaluation from here onwards.

4 Methods

This section describes the 3 ML models we initially explored through literature review and 2 additional gradient boosting solutions after our initial evaluation of their performances. From here onwards, assume that we are using the *scikit-learn* library for creating different classifiers unless specified otherwise.

4.1 Random Forest (RF)

Random Forest is a bagging method that creates an ensemble of multiple decision trees. The baseline RF model on the DrivenData website[2], with the micro F1 score of **0.5812**, provided us an initial scaffolding to quickly prototype and test better models.

One of the most relevant hyperparameters for *RandomForestClassifier* is *n_estimators*, which is the number of decision trees that will be trained inside the forest. On the available hardware of Intel Core i7 and 16GB RAM, a maximum of 500 estimators was supported without touching other hyperparameters, resulting in **0.6895**. In addition, we tried downsampling the dataset in order to have an equal number of samples in each class based on the minority class (*damage_grade*=1) as well as performing a 100-iteration *RandomizedSearchCV()* followed by a more focused *GridSearchCV()*. This, however, resulted in a worse micro F1 score at **0.6316**. Finally, an initial attempt at feature selection using *SelectKBest()* selected the 15 most relevant features, but resulted in **0.6423** using the same set of best hyperparameters found previously via *GridSearchCV()*.

4.2 Extra Trees (ET)

Extra Trees, also known as Extremely Randomized Trees, is another bagging ensemble learning approach. Similar to Random Forest, it makes predictions based on majority voting from the trees for our classification problem. But ET differs from RF in two significant ways that help reduce bias and variance. First, ET constructs trees over the entire learning sample with different subsets of features, instead of through bootstrapping. Second, ET splits the nodes randomly when it builds its decision trees.

For our base ET model, we used the DrivenData benchmark blog scaffolding and replaced it with *ExtraTreesClassifier* and achieved the initial score of **0.579**, which was similar to RF. We added

[2]https://drivendata.co/blog/richters-predictor-benchmark/

max_features to the list of hyperparameters for *GridSearchCV()*, as explored in our studied paper for ExtraTrees [5], and its resulting score was slightly better, but not significantly, at **0.5812**. This was expected considering that the paper mentioned how their ET model reported similar prediction accuracy to RF and that changing hyperparameters did not seem to improve the accuracy significantly for their Nepal dataset. Playing around with hyperparameters further and training over the entire training dataset rather than over a few selected features did increase our score to **0.633**.

4.3 XGBoost

XGBoost, which stands for Extreme Gradient Boosting, is a scalable gradient boosted decision tree algorithm built on the principles of ensemble modeling. Like other gradient boosting methods, XGBoost consists of multiple classifiers that are weak learners wherein a tree is built at each step and learns from its previous classification mistakes at the next iteration through assigned weights. Gradient boosted trees like XGBoost minimize bias and underfitting but at the same time run the risk of overfitting the model.

For our base model, we replaced the DrivenData benchmark blog scaffolding with *XGBClassifier* and got the score of **0.585**, which is comparable to the original. In order to make improvements, we used all the available features with initial hyperparameter tuning and achieved promising results with a score of **0.7459**.

XGBoost, which is a gradient boosting algorithm, resulted in better initial scores than Random Forest and Extra Trees, which are both bagging methods. Therefore, we decide to explore 2 more gradient boosting algorithms, LightGBM and CatBoost, and compare them with XGBoost, using the same initial optimal hyperparameters of *n_estimators*=200, *max_depth*=10, and *max_leaves* set to unlimited.

4.4 LightGBM

LightGBM is a gradient-boosting framework developed by Microsoft. It is quite similar to XGBoost, but the difference in its tree construction that allows much faster training. Unlike XGBoost, LightGBM does not grow a tree row-wise per level. Instead, it grows trees node-wise, by choosing the leaf that will yield the largest decrease in a given loss function [8]. When we trained our LightGBM classifier with the same hyperparameter tuning from our initial XGBoost model, we achieved the score of 0.7426, comparable to the XGBoost's score of **0.7459**.

4.5 CatBoost

Catboost is a gradient-boosting framework developed by Yandex. Its two noteworthy features are: 1) high performance without much need to tune hyperparameters, and 2) handling categorical variables through quantization, instead of one-hot encoding that we do for other algorithms. With these advantages, CatBoost prides itself to being less prone to overfitting and thus more generalizable [8]. When we trained our CatBoost classifier with the same hyperparameter tuning from our initial XGBoost model, we achieved the score of **0.7274**, which was lower than the scores of XGBoost and LightGBM.

4.6 MLExtend

During our initial investigation, we have also come across a Python library called MLExtend, which is able to combine different machine learning models and create a meta-classifier. We have explored in particular *EnsembleVoteClassifier* and *StackingClassifier* in detail to see if we can combine our base classifiers for better score. We quickly realized that using either weak base classifiers (i.e. bagging algorithms) or strong base classifiers (i.e. gradient-boosting algorithms) did not improve the F1 score when using the same features and hyperparamters for the base classifiers.

4.7 Summary

After exploring these five different ML classifier models and its combinations to create a meta classifier, our conclusions are threefold. First, gradient-boosting methods are by far superior to bagging-based approaches for our project on their own, as shown by the initial best scores for the

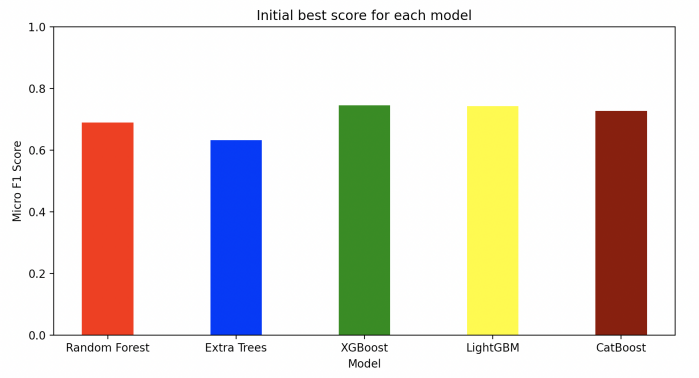

Figure 1: Initial Micro F1 Scores

5 different models in Fig. 1. Second, XGBoost and LightGBM are better fitted for our problem than CatBoost by a slight margin based on our initial attempts. Third, if we want to move forward with a combining approach using MLXtend, we need to perform hyperparameter tuning and feature selection specific to each base classifier to improve the performance.

5 Results

This section describes our different attempts of preprocessing (i.e. feature selection and engineering) and hyperparameter tuning that we performed to further improve our scores and discuss our final best solution.

5.1 Feature Selection and Engineering

Feature selection is learning and choosing specific features that will be most crucial in training our model. Feature engineering is generating new features based on combinations of existing features that will help the model training further. We describe our efforts to perform this automatically and manually.

5.1.1 Feature Selection

Besides exploring feature selection through *SelectKBest()* as we did in our initial efforts for RF, we also explored MLXtend's *Sequential Feature Selector* (SFS) to perform feature selection automatically. What this method allows is selecting separate lists of best features for different base classifiers that are then ensembled together for a final vote in MLXtend. For example, using the LightGBM classifier, SFS iteratively creates and evaluates new dataframes containing a subset of the original columns and returns the set of columns that result in the best scoring metric. After performing SFS on the one-hot encoded version of the dataset, the following features were returned.

['geo_level_1_id', 'geo_level_2_id', 'geo_level_3_id', 'age', 'has_superstructure_adobe_mud', 'has_superstructure_mud_mortar_stone', 'count_families', 'has_secondary_use', 'foundation_type_r', 'foundation_type_u', 'foundation_type_w', 'roof_type_n', 'roof_type_q', 'roof_type_x', 'position_t']

5.1.2 Feature Engineering

Based on our basic understanding of the earthquake domain, we have tried manually adding 8 new features based on the existing features on the dataset. Some examples are shown below:

- 'area_pressure' = ('count_floors_pre_eq' / 'area_percentage')
- 'old_height' = ('age' * 'height_percentage')
- 'fam_weight' = ('count_families' * 'area_pressure')

- 'all_int_plus' = ('count_families' + 'height_percentage' + 'area_percentage' + 'count_floors_pre_eq' + 'age')

However, utilizing these manually generated features did not further improve our scores, so we turned to the PyCaret library as an automated way to perform feature engineering. Using the original dataset, PyCaret was able to generate around 70 new features, which were used to train an XGBoostClassifier. The top 10 most relevant features are shown in Fig. 2.

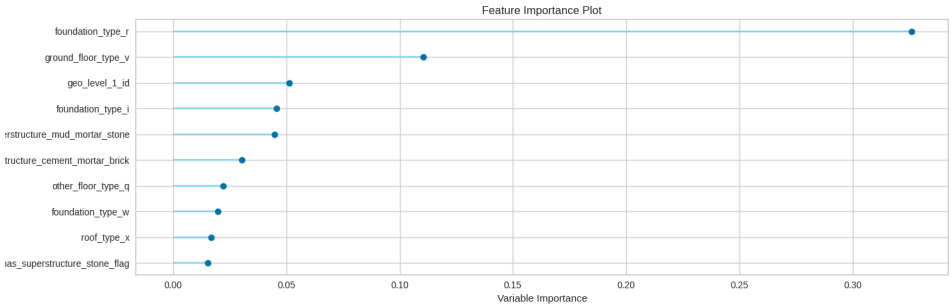

Figure 2: 10 Most Relevant Features from PyCaret

5.2 Hyperparameter Tuning

For the parameter tuning, we used a library called Optuna which is a self-contained, intelligent hyperparameter tuning library which employs efficient and intelligent randomized grid search algorithms making it a better tool than the GridSearchCV provided by the scikit-learn library.

We focused many of the hyperparameters for Catboost and XGBoost (LightGBM used the same parameter values as XGBoost since the frameworks are really similar). We tried to focus on parameters that would improve the overfitting problem that is highly possible due to our imbalanced dataset. In the end, we settled on the following parameters for our base classifiers:

- XGBoost: ['tree_method', 'n_estimators', 'max_depth', 'max_leaves', 'eta', 'subsampl'e, 'colsample_bytree', 'colsample_bylevel', 'min_child_weight', 'gamma', 'reg_lambda', 'reg_alpha']
- CatBoost: ['border_count', 'iterations', 'depth', 'grow_policy', 'learning_rate', 'min_child_weight', 'l2_leaf_reg']

Furthermore, for our ensemble model, we employ soft voting for which we need to decide the "vote share" or the "weights" that each model contributes to the final vote. For this, we used the minimize() function of the optimize method provided by SciPy to minimize the error of our ensemble. More specifically, we use the Nelder-Mead method which works in an efficient manner, especially for non-smooth functions, since it does not require any derivative information.

5.3 Final Solution

Our final solution that achieved our best micro F1 score of **0.7527** can be found in our accompanying GitHub repo[3], in the file titled 'catboost_test_final.ipynb'. The high-level details are as follows.

The feature selection and engineering methods described above have many common features related to the structure, geo_id and land characteristics. However, training the classifiers on some combination of these common features did not significantly improve the final F1 micro score. Therefore, we focused on improving hyperparameter tuning over the entire dataset.

In terms of hyperparameter tuning, we optimized them using Optuna for each base classifier as mentioned in the previous section. Using these as the base classifiers, we created a weighted

[3]https://github.com/briankim113/CSCI567_FinalProject

ensemble of the three gradient-boosting methods (XGBoost, LightGBM, and CatBoost), where the optimal weights were [0.95991211, 0.99814453, 1.0], respective to each model. This was calculated using the scipy.optimize() function as described above.

6 Conclusion

This paper outlines our progress from literature review and initial model evaluation to data preprocessing and hyperparameter tuning. Our final best micro F1 score of **0.7527** is achieved by creating a meta-classifier that takes in the "weighted votes" from our three base classifiers (XGBoost, LightGBM, CatBoost), each with their own optimal hyperparameters, to get our final predictions.

References

[1] Kuldeep Chaurasia, Samiksha Kanse, Aishwarya Yewale, Vivek Kumar Singh, Bhavnish Sharma, and B. R. Dattu. Predicting damage to buildings caused by earthquakes using machine learning techniques. In *2019 IEEE 9th International Conference on Advanced Computing (IACC)*, pages 81–86, 2019. doi: 10.1109/IACC48062.2019.8971453.

[2] Weiyi Chen and Limao Zhang. Building vulnerability assessment in seismic areas using ensemble learning: A nepal case study. *Journal of Cleaner Production*, 350:131418, 2022. ISSN 0959-6526. doi: 10.1016/j.jclepro.2022.131418.

[3] Anmol Gaba, Arnab Jana, Rahul Subramaniam, Yash Agrawal, and Merin Meleet. Analysis and prediction of earthquake impact-a machine learning approach. In *2019 4th International Conference on Computational Systems and Information Technology for Sustainable Solution (CSITSS)*, pages 1–5, 2019. doi: 10.1109/CSITSS47250.2019.9031026.

[4] Subash Ghimire, Philippe Guéguen, Sophie Giffard-Roisin, and Danijel Schorlemmer. Testing machine learning models for seismic damage prediction at a regional scale using building-damage dataset compiled after the 2015 gorkha nepal earthquake. *Earthquake Spectra*, 38(4):2970–2993, 2022. doi: 10.1177/87552930221106495.

[5] Ehsan Harirchian, Vandana Kumari, Kirti Jadhav, Shahla Rasulzade, Tom Lahmer, and Rohan Raj Das. A synthesized study based on machine learning approaches for rapid classifying earthquake damage grades to rc buildings. *Applied Sciences*, 11(16):7540, Aug 2021. ISSN 2076-3417. doi: 10.3390/app11167540.

[6] Sajan K C, Anish Bhusal, Dipendra Gautam, and Rajesh Rupakhety. Earthquake damage and rehabilitation intervention prediction using machine learning. *Engineering Failure Analysis*, 144: 106949, 2023. ISSN 1350-6307. doi: 10.1016/j.engfailanal.2022.106949.

[7] Aishwarya Kumaraswamy, Bhargava N Reddy, and Rithvik Kolla. Richter's predictor: Modelling earthquake damage using multi-class classification models. In *2020 Third International Conference on Advances in Electronics, Computers and Communications (ICAECC)*, pages 1–5, 2020. doi: 10.1109/ICAECC50550.2020.9339484.

[8] Andre Ye. Xgboost, lightgbm, and other kaggle competition favorites, 2020. URL `https://medium.com/analytics-vidhya/xgboost-lightgbm-and-other-kaggle-competition-favorites-6212e8b0e835`.