Team20 - Richter's Predictor: Modeling Earthquake Damage

Shaonan Hua
Department of Computer Science
University of Southern California
Los Angeles, CA 90007
`shaonanh@usc.edu`

Zihan Shen
Department of Computer Science
University of Southern California
Los Angeles, CA 90007
`zihanshe@usc.edu`

Rucheng Zhou
Department of Information System Engineer
University of Southern California
Los Angeles, CA 90017
`ruchengz@usc.edu`

Abstract

The goal of the competition was to predict the damage level of buildings following the 2015 Gorkha earthquake based on their features. Our team focused on improving prediction accuracy through both data and model enhancements. To start, we explored various feature engineering techniques to enhance the data. On the model side, we tested several models, adjusted hyperparameters, and utilized ensembling methods to achieve the best results. We also implemented cross-validation to prevent overfitting. In the end, our efforts resulted in a final F1 score of 0.7486.

1 Introduction

The 2015 Gorkha earthquake in Nepal caused significant damage to buildings, with three levels of damage (1-3) observed. In this project, we aim to predict the level of damage to buildings using data collected through surveys by Kathmandu Living Labs and the Central Bureau of Statistics, which works under the NPCS of Nepal. The dataset contains 260,601 datapoints and 41 features, including geographic position, age and height of buildings, and the material used in construction. We plan to use data mining and machine learning techniques to predict the level of damage and achieve a precise F1-score.

2 Methods

2.1 Data preprocessing

The first step in our analysis was data preprocessing. The dataset contains 41 features, including 1 meaningless feature called building id, 8 categorical features, and 32 numerical features.

Numerical feature Among 32 numerical features, there are 3 numerical features called "geo level 1, 2, 3" did not contain numerical meanings and 24 binary features. So, we picked up the remaining 5 features ('count floors pre eq', 'age', 'area percentage', 'height percentage', 'count families') as true numerical features and applied log transformation plus standardScaler to make these features have normal distributions.

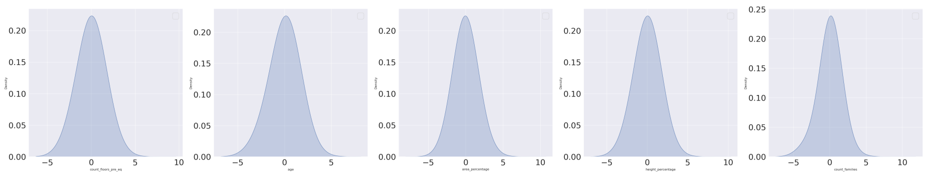
Figure 2: Numerical Features after preprocessing

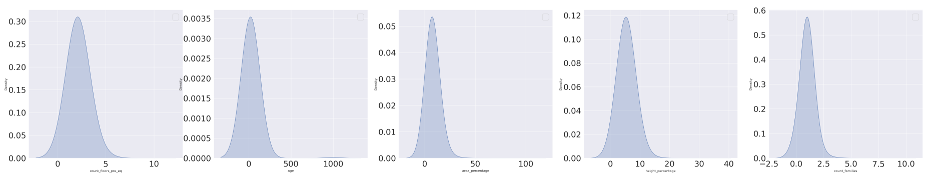
Figure 1: Numerical Features before preprocessing

For the geo levels, different levels have different value ranges, the range of level 1 is 0 to 30, level 2 is 0 to 1427 and level 3 is 0 to 12567. For each geo level, we grouped by the data according to the geo level, then calculated the damage grade's mean, counts, and standard deviation for the corresponding geo level value, and saved the results into dictionaries. Then we added three columns named tar_enc_geo_level_mean, tar_enc_geo_level_count and tar_enc_geo_level_std into the dataset based on the dictionaries value. After this process, we added 9 columns.

Categorical feature For categorical features, we applied ordinal encoding, one-hot encoding and target encoding. Ordinal encoding and onehot encoding didn't make any improvements, so we chose target encoding as our final solution for categorical features. When applying target encoding for the categorical column, we only computed the damage grade's mean for each column value, so we added 5 more new columns.

Feature Combination Besides, according to the feature importance, we noticed that certain values of object column had significant importance. Thus, we tried to combine two different features as a new feature, like foundation type and roof type, foundation type and floor type, foundation type and position, groud floor and other groud floor, (Figure 3, 4, 5, 6) etc. However, it did not give a more precise prediction.

PCA Since our dataset contained up to 81 features after encoding, we explored the use of principal component analysis (PCA) to reduce the dimensionality of the data. PCA is a common technique used to simplify complex data by identifying patterns and relationships between features. After applying PCA, we examined the first 20 features of the first component to determine their importance in predicting building damage levels. However, this process did not result in a significant improvement in accuracy, so we decided to keep all features for the final process.

Although using all features can sometimes result in overfitting or slower processing times, we determined that keeping all features was the best approach for this specific problem. By retaining all features, we were able to capture as much information as possible about each building's characteristics and their relationship to the predicted damage levels.

We then inspected the distribution of the dataset and found that the numerical columns had a right-skewed distribution, with varying ranges. To address this issue, we applied standardization and normalization. Additionally, the damage grade had a highly unbalanced distribution, with most buildings falling into the third damage grade. There are 148259 buildings that belong to damage 1, 87218 belong to damage 2, and only 251254 belong to damage 3. The unevenness reminds us that the weight of each damage should be adjusted when we are looking for the hyperparameter of model. Our team realize it is very important to choose a model that is robust to unbalanced dataset.

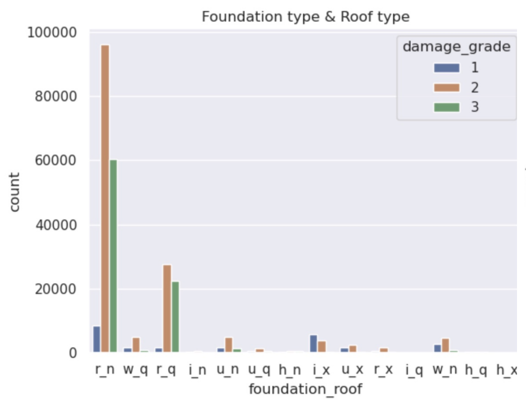

Figure 3: Foundation type and roof type

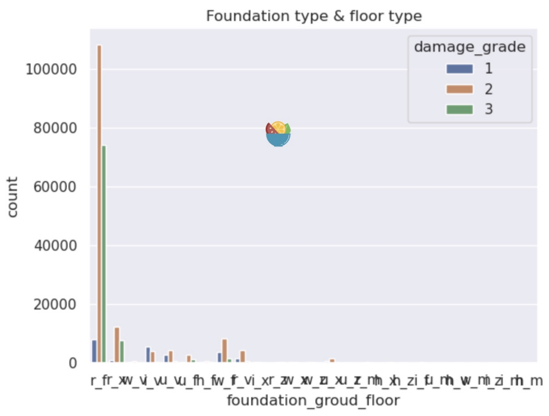

Figure 4: Foundation type and floor type

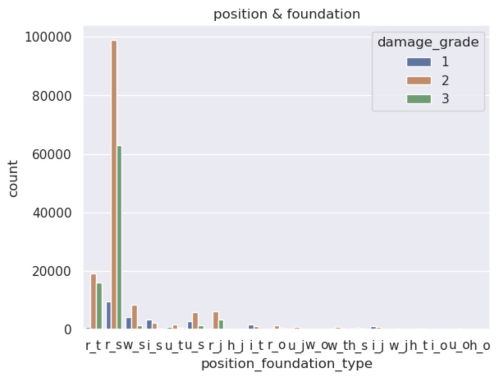

Figure 5: Foundation type and position

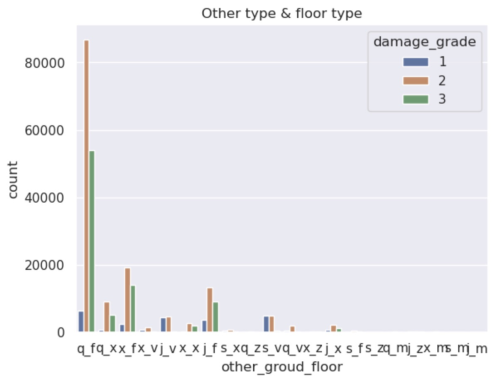

Figure 6: Groud floor and other groud floor

2.2 Model Selection

After pre-processing the dataset, we tried four single classification models to predict the extent of damage. We used grid search and 5-fold validation to find the best hyperparameters for each model.

Random Forest Random forest is commonly used in classification problems, which can be more accurate than decision tree algorithm. Therefore, random forest can be used as a good initialization model. The parameters of our model are as follows.

Parameters	Value
n_estimators	200
max_depth	10

XGBoost XGBoost is a complex model compared to the random forest, and usually performs better than RF when there is a class imbalance.

Parameters	Value
n_estimators	1000
learning_rate	0.1
max_depth	5

LightGBM For Lgbm, the best hyperparameters were "num_leaves," "learning_rate," "max_depth," "lambda_l2," and "max_bin." By setting "is_unbalance" to "True" to address the imbalanced dataset, we achieved a validation set F1-score of 0.7527 and a training set F1-score of 0.8231.

Parameters	Value
n_estimators	1400
num_leaves	230
learning_rate	0.1
lambda_l2	5
max_bin	90
max_depth	30
is_unbalance	True

CatBoost For Catboost, the best hyperparameters were "n_estimators," "learning_rate," "l2_leaf_reg," and "max_depth." By setting the hyperparameter "class_weights" to "[1,2,2.5]" to balance the data, we achieved a validation set F1-score of 0.752.

Parameters	Value
n_estimators	1500
learning_rate	0.1
l2_leaf_reg	10
max_depth	7
class_weights	[1,2,2.5]

2.2.1 Model Comparison

Model	F1-Score
Random Forest	0.7341
XGBoost	0.7422
LightGBN	0.7445
CatBoost	0.7448

As can be seen from the table, the accuracy using a single random forest model was relatively low at 73.41%. XGBoost performed better than the traditional random forest on this dataset with an accuracy of 74.22%. We then tried two other gradient boosting algorithms Lightgbm and CatBoost, both of which improved the classification accuracy over XGBoost. Up to this point, a classification accuracy of 74.48% could be achieved using a single model, with the catboost model performing the best.

2.3 Ensemble Methods

Ensemble methods aim at achieving better results in machine learning by combining multiple models instead of using a single model. It helps to improve the robustness of the model by viewing a group of models as a whole rather than individuals. In this report, we implement the Max Voting as the basic ensemble method and then chose Stacking as an advanced solution.

Max Voting Max Voting is mainly used for classification problems, which consists of building several models independently and getting all their outputs called 'vote'[1]. In the case of classification, the predictions for each sample are depend on the majority vote.

There are two approaches to the max voting for classification: hard voting and soft voting. Hard voting outputs the label with the largest sum of votes from models. A slight different from hard voting, soft voting try to sum the probabilities for each label and predict the class with the largest probability. Usually soft voting training is more effective, so the ensemble methods of soft voting is used in this training process.

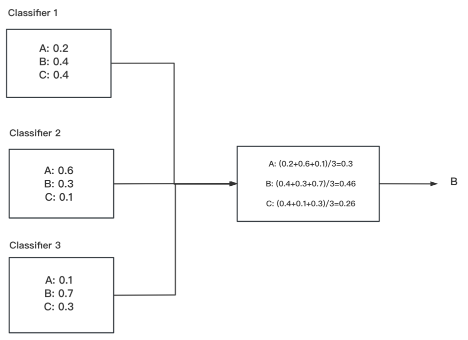

Figure 7: Soft Voting

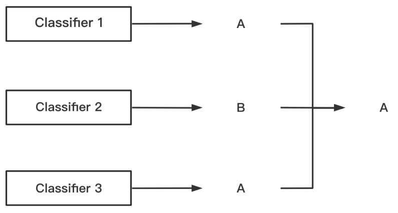

Figure 8: Hard Voting

To implement max voting method, three classification models(XGBoost, LightGBM, CatBoost) were combined by using the VotingClassifier, which was trained and outputs the class with the maximum probability.

Stacking Stacking is the other ensemble method that combines multiple models via a meta-model. It has first-level and second-level models. Some features are extracted by training the base-model on the whole training dataset[1]. The meta-model is then trained by using those features and finally output the test predictions.

In this project, we tried two stacking approaches to ensemble models. For the first approach, we chose LightGBM and XGBoost as the base models. After loading the first-level learning algorithms, we used XGBoost as the second-level model to be fitted with stack features. Then we predicted the final predictions using stacking. In the second approach, we used CatBoost and XGBoost as the base model, then used logistic regression to combine the predictions of the sub-models.

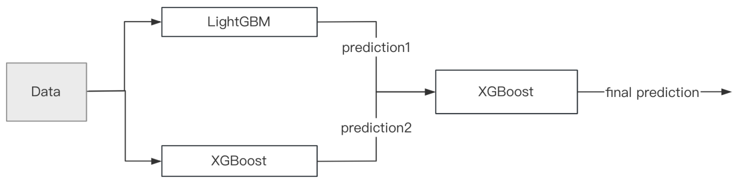

Figure 9: Stacking method 1

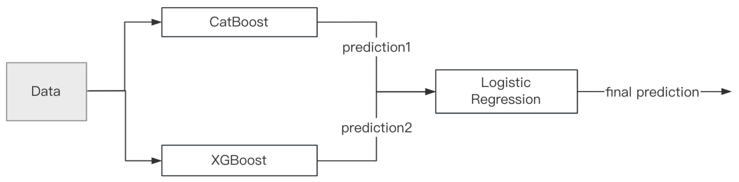

Figure 10: Stacking method 2

3 Results

3.1 Results of approaches

As mentioned before, the accuracy that can be achieved using a single CatBoost model is 74.48%. After using the basic ensemble approach, the classification accuracy was improved to 74.86%. In addition to this, we tried two different stacking methods, but neither of them showed any significant improvement in classification results.

3.2 Analysis

In terms of the results, combining multiple models gives the ensemble model a stronger generalisation capability. The classification accuracy of the ensemble model increased significantly compared to the use of a single model.

There are several possible reasons for the lack of improvement in the classification accuracy of the models after using stacking:

1. The accuracy of the model in the first-level is low. We can later try the case of using different base models again and try to choose the classifier with high accuracy.
2. The number of base classifiers is small. In the two methods tried above, we only use two models in the first layer. In the future, we can try to increase the number of base models or increase the richness of the base models by changing the parameters.
3. The features capture in the first layer is missing. An attempt could be made to include the original data in the second layer.

In conclusion, we used data mining and machine learning techniques to predict the extent of damage to buildings caused by the 2015 Gorkha earthquake in Nepal. We performed data pre-processing, model selection and stacking to achieve an F1-score of 0.7486.

References

[1] Apurva_007 (2023, March 27) Ensemble Methods in Python from https://www.geeksforgeeks.org/ensemble-methods-in-python/

A Appendix

A.1 Details on how to run the code

Use Python3.
Use GPU as the accelerator.

Package installation If you use Kaggle, no need to install any packages.
Otherwise, please install the following packages in your python environment:
pandas, numpy, matplotlib, seaborn, sklearn, xgboost, lightgbm, catboost, vecstack.

Data Path Set your data path in the variable **input_path**, and set the variable **output_path** as where you want to save your submission file.