Final Report for Richter's Predictor: Modeling Earthquake Damage

Enyu Zhao
Department of Computer Science
University of Southern California
Los Angeles, CA 90089
`enyuzhao@usc.edu`

Yanjun Gu
Department of Computer Science
University of Southern California
Los Angeles, CA 90007
`yanjungu@usc.edu`

Abstract

The report presents an algorithmic approach to tackle the Richter's Predictor: Modeling Earthquake Damage problem, achieving an impressive f1-score of 0.7525. The algorithm incorporates a range of data preprocessing techniques, including feature selection based on mutual information, log transformation to normalize numerical features, geological feature encoding using a neural network embedding, and Principal Component Analysis to reduce the dataset's dimensionality. LightGBM is selected as the primary training model following a rigorous evaluation of its performance compared to other models such as Neural Network, XGBoost, and Catboost. Lastly, Ensemble learning is utilized to combine the predictions of multiple models, thereby enhancing the algorithm's generalization ability.

1 Algorithms Used

As one of the inevitable and unpredictable natural catastrophes, strong earthquakes are horrifying and devastating, shattering houses into ruins and leaving people moaning about their loss. Therefore, using well-developed machine learning algorithms to predict the damage grade of the houses after the earthquake is essential. Predicting the status of the house after the terrifying earthquake can be beneficial for the government as the search and rescue teams can be efficiently dispatched to the most needed houses and areas at the earliest possible time right after an earthquake was detected. In the long term, government and insurance companies can exploit the prediction for better risk assessment and insurance pricing.

With LightGBM [1], XGboost [2] and Catboost [3] being the most popular machine learning algorithms in classification, we will use them to predict the damage grade. With neural networks' dominant place in today's machine learning algorithms [4], we will also give it a try.

2 Data Preprocessing

2.1 Mutual Information

Mutual Information measures the mutual dependence between two random variables and is commonly used in machine learning as a feature selection technique. By quantifying how much information one variable can provide about another, it identifies variables with higher relevance to the target variable. One of the significant advantages of Mutual Information is its versatility and robustness. It can handle both discrete and continuous variables and is relatively insensitive to noise and outliers in the data. However, it may not always be the optimal feature selection technique for a specific problem and can be influenced by the curse of dimensionality in high-dimensional datasets.

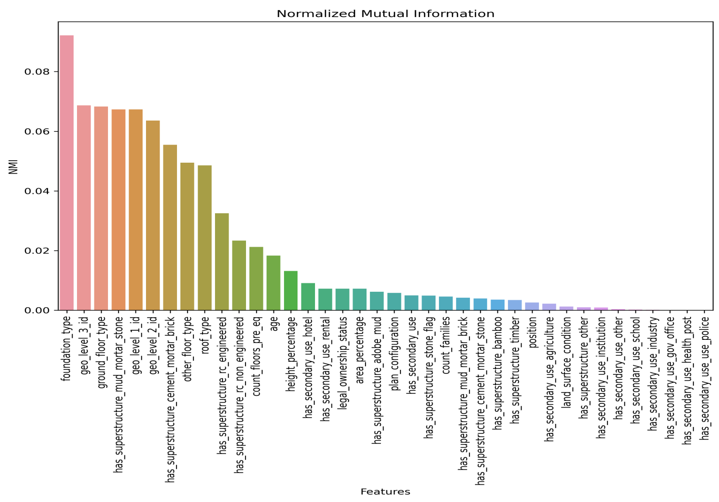

Figure 1: Normalized mutual information of features.

The normalized mutual information of all features is shown in Figure 1. We select the features whose normalized mutual information > 0.001 as training features.

2.2 Log Transformation

Skewed data in machine learning refers to datasets where the distribution of values in a feature or target variable is heavily skewed towards one end of the range. Skewed data can create challenges for machine learning practitioners, including biased models and poor generalization. Machine learning models may become biased towards the majority class, leading to poor performance on the minority class. Skewed data can also make it difficult for models to generalize to new data. Log transformation is a powerful technique for addressing skewed data in machine learning. By applying a logarithmic function, log transformation can reduce skewness and make the data more normally distributed. This can help to overcome challenges such as biased models and poor generalization by reducing the potential for bias towards the majority class in the model. Overall, log transformation is an essential tool for enhancing the accuracy and performance of machine learning models when dealing with skewed data.

Figure 2 displays the Kernel Density Estimate plots of numerical features before and after log transformation. We applied this transformation to our data and saw an improvement in the f1-score of our machine learning models. Notably, we observed an increase in the F1 score of the Random Forest, XGBoost, and Catboost models from 0.7264 to 0.7322, from 0.7356 to 0.7359, and from 0.7416 to 0.7421, respectively.

2.3 Feature Embedding

During the process of feature selection using mutual information, we observed a high degree of interdependence among the 3 hierarchical geologic features, as well as their significant influence on the target variable, i.e., the damage grade.

However, the acquired data presents the important geologic features only as number identifiers, which lack the necessary information for determining the exact location of a house. To extract and utilize the latent information concealed within these features, we employed an autoencoder for encoding the geologic data. The architecture of the autoencoder model utilized in our study is presented in Figure 3.

An autoencoder is a type of neural network that is used for unsupervised learning, dimensionality reduction, and feature extraction, Hinton G E, et al. [5]. The main objective of an autoencoder is

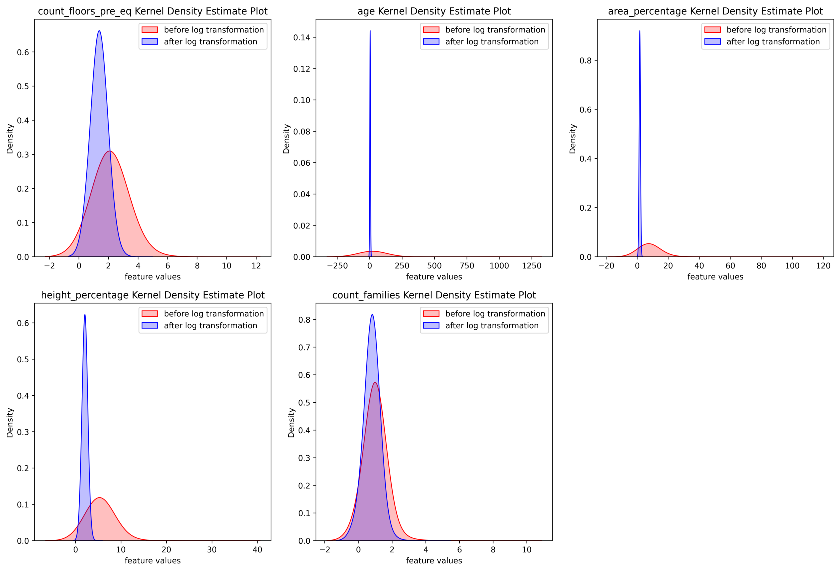

Figure 2: Kernel Density Estimate plots of numerical features.

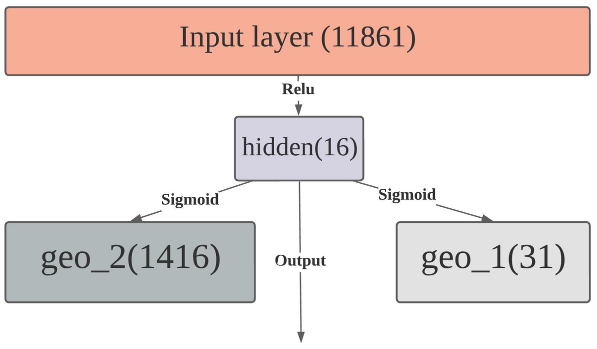

Figure 3: Structure of autoencoder.

to learn a compressed representation of input data by encoding it into a latent space with different dimensions and then decoding it back into the required form. In our study, we want to transform the 3 features with different layers of geological identifiers into 16 features.

Following the utilization of the autoencoder, our subsequent LightGBM and neural network models demonstrated significant performance improvements.

2.4 Principle Component Analysis

Principal Component Analysis (PCA) is a multivariate statistical technique commonly used in data science and machine learning to reduce the dimensionality of large datasets while retaining as much of the original variance as possible, Shlens J, et al [6]. PCA is based on the identification of the principal components in the data, which represent linear combinations of the original variables that capture the maximum amount of variation in the dataset. These principal components are chosen such that each one is orthogonal to the others, ensuring that they are independent and capture unique aspects of the underlying data structure.

In an effort to simplify the dataset without sacrificing valuable information, we employed the use of Principal Component Analysis (PCA) to perform dimension reduction on the non-geological features.

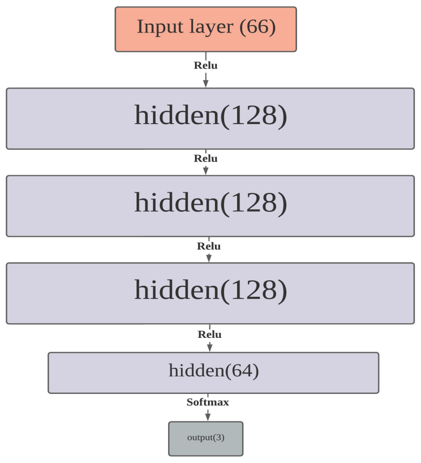

Figure 4: Structure of neural network for classification.

Although our results showed that this approach did not yield a significant improvement in our model's performance, it notably enhanced the training speed of our models.

3 Model training

In our experiment, we first established a baseline benchmark by training both a neural network model and a LightGBM model without any data preprocessing techniques. Subsequently, we utilized our proposed preprocessing methods to train these models again, with the aim of evaluating the impact of the preprocessing on the models' performance. Finally, we trained all models on the fully preprocessed dataset in order to identify the model with the highest level of performance. After data preprocessing and hyper-parameter tuning, we gained the results listed in Table 1.

3.1 Neural Network Training

Prior to implementing any preprocessing techniques on the given dataset, we established a baseline benchmark by utilizing a neural network architecture that incorporated all available features. The network architecture is presented in Figure 4.

Upon evaluation of the neural network model with optimal hyperparameters on the test set, the f1-score was 0.6614. This score fell significantly below the level of performance deemed acceptable, indicating that the neural network model requires further optimization in order to achieve satisfactory results.

A potential optimization method is to utilize the autoencoder. We assessed the efficacy of such method, which led to an improvement in performance from 0.6614 to 0.7213. Additionally, we implemented log transformation and feature selection techniques, which resulted in a further improvement in the neural network's performance to 0.7301.

3.2 LightGBM Model Training

We initially employed LightGBM as our classification algorithm while concurrently exploring the use of neural network models as a benchmark. It was observed that the performance of the LightGBM model was superior to that of the neural network model as shown in Table 1.

Following the implementation of geological feature embedding and subsequent training of the LightGBM model on the processed dataset, an improvement in performance was observed, with the f1-score increasing from 0.7237 to 0.7416. Further improvement was attained through the application of log transformation and feature selection, which resulted in a performance of f1-score reaching 0.7421.

Table 1: Model result

Algorithm	Original f1-score	Preprocessed f1-score	Hyper-parameter tuned
Neural Network	0.6614	0.7301	Null
LightGBM	0.7237	0.7450	0.7490
XGboost	0.7356	0.7432	0.7456
Catboost	0.7416	0.7442	0.7483

In order to improve the performance of our LightGBM model, we conducted a hyperparameter tuning using cross-validation and grid search techniques. This process allowed us to optimize the parameters of the model and ultimately resulted in a higher f1-score of 0.7490.

3.3 XGboost Model Training

After exploring the LightGBM decision tree algorithm, we decided to experiment with XGBoost, another popular framework for classification tasks. Our initial implementation involved applying only log transformation, which yielded an f1-score of 0.7359. However, we were able to improve this score to 0.7432 by encoding the geological data with embeddings.

We then conducted a hyperparameter tuning process using grid search, resulting in further improvements to the model's accuracy. Our final f1-score of 0.7456 was achieved using a learning rate of 0.2, 40 leaves, and a maximum depth of 32.

While our experiments with XGBoost showed improved results, the algorithm's performance did not match that of LightGBM. Although we achieved an f1-score of 0.7456, we chose not to invest further resources into it, as it was not as effective as our primary model.

3.4 Catboost Model Training

Initially, we normalized all numerical data, but this led to a 0.03 decrease in the f1-score. We then applied log transformation to address the data's skewness, resulting in a marginal increase in the f1-score from 0.7416 to 0.7421.

Subsequently, we investigated the impact of incorporating geological features into the model. We first tried encoding geological data as categorical features in the Catboost model, which yielded an f1-score of 0.7436. Alternatively, we leveraged geological embedding, which proved to be more effective, leading to an f1-score of 0.7442. We also attempted to simultaneously add geological features as categorical data and apply geological embedding, but this approach fared poorly on the testing set despite achieving a high training set f1-score of 0.7480. Consequently, we focused solely on applying geological embedding.

To optimize the model, we conducted a grid search, which resulted in an f1-score of 0.7483 using a border count of 17, depth of 10, L2 leaf regularization of 5, and a learning rate of 0.07. In conclusion, Catboost proved to be a competitive alternative to LightGBM for our dataset owing to its efficient handling of categorical features and its ability to integrate geological information through embedding.

4 Further Improvement

Following the stagnation of the performance of the existing models with a maximum f1-score of 0.7490, we introduced novel techniques to improve the predictive capabilities of our models.

4.1 Ensemble learning LightGBM models

In order to further improve the performance of our model, we utilized a technique known as ensemble learning. Ensemble learning works by combining multiple individual models into a single, more powerful model that outperforms any of the individual models alone, Sagi O, et al. [7]. The intuition behind this is that the individual models may have different strengths and weaknesses, and by combining them, the weaknesses of one model can be compensated for by the strengths of another.

Table 2: LightGBM Ensemble Performance

Model number	Hard-vote f1-score	Soft vote f1-score
3	0.7512	0.7515
5	0.7520	0.7525
7	0.7518	0.7518
15	0.7517	0.7517

Table 3: Multiple Model ensemble Performance

Model Ensembled	Hard-vote f1-score	Soft vote f1-score
LightGBM,Catboost,XGboost	0.7504	0.7505
LightGBM,Catboost	0.7498	0.7498
LightGBM,XGboost	0.7446	0.7447
XGboost,Catboost	0.7442	0.7445

This leads to a more robust and accurate model that is less prone to overfitting and can generalize better to unseen data.

Ensemble learning can also reduce the impact of noise and outliers in the data. By combining the predictions of multiple models, the noise and outliers are less likely to be included in the final result, as they may be filtered out by some of the individual models.

In practice, we first split the given dataset into 5 folds and trained 5 separate LightGBM models, each using a unique fold as the test set and the other 4 folds as the training set. We then performed a hard voting scheme among all 5 models to classify the final test dataset for submission. This approach helps to reduce overfitting and improve the generalization of the model. Additionally, ensemble learning has been shown to often outperform single models, which was the case in our experiments.

An alternative method to gain the final prediction is to perform a soft vote. This involves obtaining probability estimates for each label from each model, and then aggregating these probabilities to determine the final label prediction.

In our experiments, we applied both voting methods to our models and observed improved performance compared to individual models. We conducted tests with various numbers of models to be trained and the results are presented below.

4.2 Emsemble different models

During our experimentation with ensemble learning, we found that combining multiple classification algorithms using hard vote or soft vote could further improve the performance of the models. Similar to the process of ensembling lightGBM models, we partitioned the dataset into K folds, where K represents the total number of models from all algorithms. We then applied hard vote and soft vote techniques to combine the predictions from all models and obtain the final classification with the result shown in Table 3.

From the result we noticed that combing all algorithms' predictions produces worse performance, it is hypothesized that the poor performance of some algorithms may have a negative effect on the "hard" examples, which are those instances where the probability of the correct label is only slightly higher than the probabilities of the wrong labels.

In addition to ensembling all algorithms together, we also attempted to ensemble the models from each pair of algorithms. However, our experiments showed that this approach did not improve the overall performance either.

References

[1] Ke G, Meng Q, Finley T, et al. *Lightgbm: A highly efficient gradient boosting decision tree[J]. Advances in neural information processing systems*, 2017, 30.

[2] Chen T, Guestrin C. *Xgboost: A scalable tree boosting system*[C]//Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining. 2016: 785-794.

[3] Prokhorenkova L, Gusev G, Vorobev A, et al. *CatBoost: unbiased boosting with categorical features[J]. Advances in neural information processing systems*, 2018, 31.

[4] Abiodun O I, Jantan A, Omolara A E, et al. *State-of-the-art in artificial neural network applications: A survey*[J]. Heliyon, 2018, 4(11): e00938.

[5] Hinton G E, Salakhutdinov R R. *Reducing the dimensionality of data with neural networks*[J]. science, 2006, 313(5786): 504-507.

[6] Shlens J. *A tutorial on principal component analysis*[J]. arXiv preprint arXiv:1404.1100, 2014.

[7] Sagi O, Rokach L. *Ensemble learning: A survey*[J]. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, 2018, 8(4): e1249.

Appendix

Code instruction

We tested our code on macOS and Windows, it should be noted that macOS with Apple Silicon chips including M1, M1 pro, M1 Max and M2 won't support LightGBM so the code should be run on Windows.

To run our code, the operating folder should be the outer folder itself. Suppose the whole folder is located at *C:/Codes/ FinalProj*, then in conda or the terminal, you can enter the folder by using *cd* command to *C:/ Codes/ FinalProj*.

The command to run our code are listed below for different scenarios:

To run neural network:

```
python main.py --y-onehot --epochs=100 --evaluate
```

As we didn't perform any ensemble mechanism on the neural network, you don't need to set *softvote* parameter which will be set in the following models.

To run LightGBM with hard vote and train 5 models to ensemble:

```
python main.py --algorithm=lgbm --no-y-onehot
--no-evaluate --no-softvote --retrain --ensnum=5
```

The model files will be stored in *Finalproj/[algorithm]method/model/training_time*, in this case,it will be stored in *Finalproj/lgbm_method/model/training_time*. If you don't want to retrain the models, you can take all the models stored in the *training_time* folder to the *[algorithm]method/model* folder. Feel free to toggle around with *algorithm, retrain, softvote* as we will show some examples below. The *ensnum* parameter however, is only supported in LightGBM.

To run catboost with softvote and retrain:

```
python main.py --algorithm=cat
--no-y-onehot --no-evaluate --softvote --retrain
```

To run xgboost with softvote and no retrain:

```
python main.py --algorithm=cat --no-y-onehot
--no-evaluate --softvote --no-retrain
```

To run an ensemble of different models with softvote:

```
python main.py --algorithm=ensemble --no-y-onehot
--no-evaluate --softvote --retrain
```

It's suggested to run the ensemble algorithm with *retrain* set to be true since the model that you want to put in may not be pretrained. And to choose which models to ensemble, you can go to the */Finalproj/ensemble_runner/ensemble_runner.py* to uncomment some choices in the *run* function.