Modern Approaches for Modeling Earthquake Damage

AVALLE Dario
EURECOM
me@darioavalle.it

FONTANA Umberto
EURECOM
fontana@eurecom.fr

TOUAM Mohamed Achraf
EURECOM
touam@eurecom.fr

***Abstract*—This project is inspired by the competition on DrivenData about the Gorkha (Nepal) earthquake[1]. In April 2015, this earthquake, with a magnitude of 7.8 on the Richter scale killed 8964 people and left hundreds of thousands homeless with entire villages flattened. We decided to address this matter by developing a few machine learning algorithms enabling the determination of the damage grade (low, intermediate, or fully destroyed) of the houses affected by the earthquake.**
These algorithms use a dataset containing various information about the houses from their geographical position to the materials used in their construction. Therefore, these algorithms may be useful when performing pre-construction earthquake-proof quality determination. In other words, they could enable us to enhance the construction work and potentially save lives. The project GitHub is available at https://github.com/Terdis/Modeling-Earthquake-Damage.

I. Introduction

On April 25, 2015, a devastating earthquake struck Nepal[2], causing widespread damage and loss of life. The 7.8 magnitude earthquake, along with several powerful aftershocks, destroyed or damaged thousands of buildings, homes, and infrastructures, leaving many people homeless and without basic necessities.

The problem of predicting earthquake damage to buildings is a critical one, as it can help emergency responders and city planners make informed decisions in the aftermath of a disaster. In fact, a massive effort was undertaken to collect data on the affected buildings and structures. This data was collected and compiled into an extensive dataset, which has been used for a variety of research and analysis purposes.

One important use of this dataset has been to train machine learning algorithms to predict the damage caused by earthquakes. The goal of this report is to provide a detailed analysis of these algorithms, and a discussion of the results and potential applications of the models developed.

In order to develop a robust model, some basic preprocessing steps have been applied. These procedures were different from each algorithm that has been used for the classification task. Later on, we created an ensemble method between the best-performing algorithms and it relies on the estimated probability of each model. Along with some classic machine learning algorithms, we made use of some more modern boosting techniques, like the Light Gradient Boosting Machine and the Categorical Boosting. A brief description of these algorithms is also present in the model section of this report.

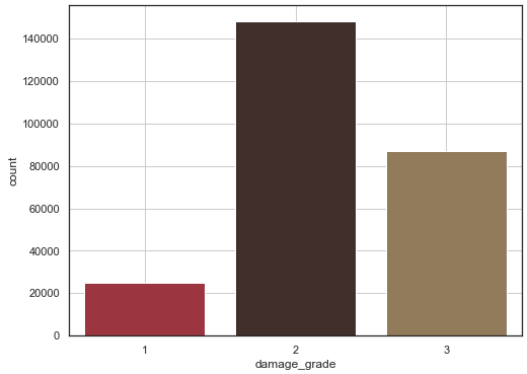


Fig. 1: Damage grade frequency

II. Dataset and features

The dataset includes 38 different kinds of information on 260601 buildings, such as the age of the building, the number of floors, and the type of construction. The task is a classification one, where the label set is composed of three possible values, which are the following:

1) low damage
2) medium damage
3) complete destruction

The training set is imbalanced as label 2 is much more frequent than the others, as shown in Figure 1.

We didn't have to correct any data: there is no null value, and the semantics of the features is respected (i.e. the age dimension always assumes positive values).

The heatmap in Figure 2 shows that not many features are highly correlated. The highest correlation values revealed to be between the height percentage and the number of floors (as expected since the higher the building is, the higher the number of floors of the building), with a correlation of 0.77, and between the features *"has_secondary_use_agriculture"* and *"has_secondary_use"*, with a correlation of 0.74. These values though are not high enough to consider some features as redundant and perform a Principal Component Analysis or any other feature reduction technique.

In the outlier detection, we discovered that there are 1390 samples having ages equal to 995, while every other building has an age lower than 200 (Fig. 3).
We made the hypothesis that those buildings are outliers, even if we discovered that they really exist and were effectively damaged by the earthquake[3].

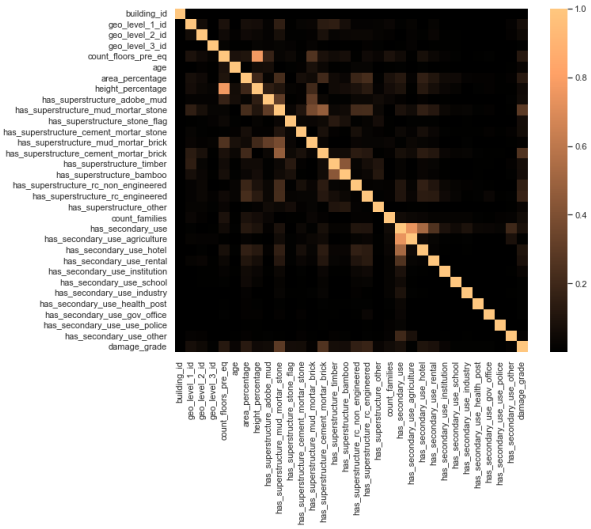


Fig. 2: Correlation map between the features

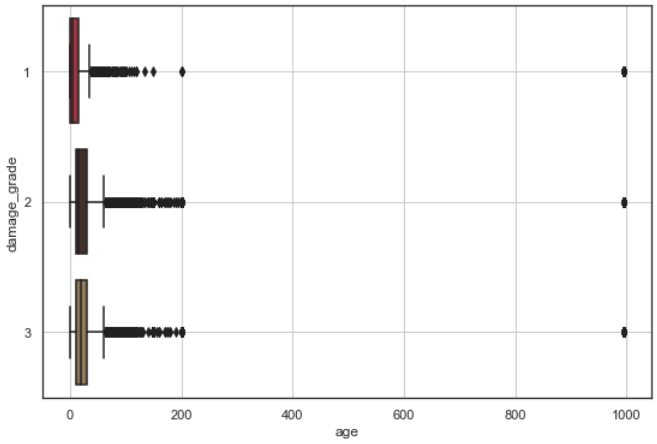
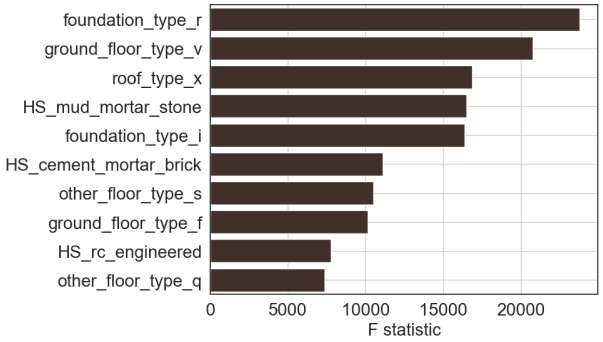


Fig. 3: Boxplot of the age feature

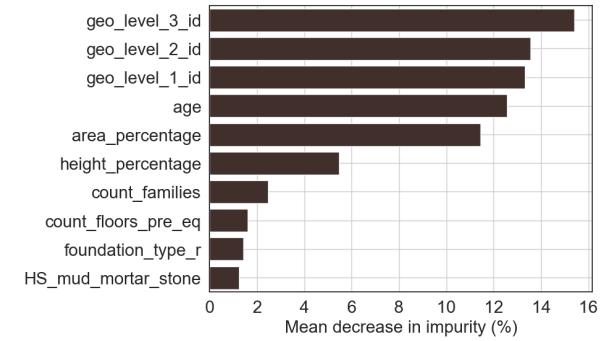
We ran some selection algorithms to figure out which are the most relevant features.

- *ANOVA*: Analysis of Variance [4] is a method that uses an F-test to determine a feature's importance. It makes the null hypothesis that a given feature is useless in explaining the linear model that correlates data with their labels. It then compares the variance of the model with and without that feature and returns the F-value: if it is high, the null hypothesis cannot be accepted and this means that the feature is important.
- *Random Forest Classifier*: the random forest implemented in scikit learn has a property called *feature_importances_*, which returns a pandas Series whose entries represent the importance of a given feature. The importance of a feature is computed as the mean of the impurity decrease within each tree.

As show in Figure 4, according to ANOVA, the five most important features are *foundation_type_r, ground_floor_type_v, roof_type_x, has_superstructure_mud_mortar_stone* and *foundation_type_i*, while according to Random Forest, the best ones are *geo_level_id_1, geo_level_id_2, geo_level_id_3, age* and *area_percentage*. Anyway, we did not perform any feature selection since there are only 66 features in the dataset (this number is taking into account the one hot encoding).



(a) ANOVA



(b) Random Forest

Fig. 4: Most important features according to ANOVA (a) and Random Forest (b)

III. Methods

Preprocessing

Different data preprocessings have been implemented for different models but all of them are a combination of some of the following steps:

- *Compression of binary features* - since some features share the same meaning, we created a new dimension which is the sum of those features (i.e. there are 11 binary features about the superstructure of the building, and the new feature is the number of different superstructures);
- *One-Hot Encoding* - some categorical features have low cardinality. Therefore, for those features, the one-hot encoding performs well without increasing too much the number of dimensions of the data;
- *Target Encoding* - The geographical position of the building is denoted by the three geo-level features, where each feature denotes a more enclosed geographical region. At first look, those features might seem numerical, but better results come up if they are considered categorical since they don't represent any measure, but they are identifiers

indeed. If we consider them as categorical, though, a one-hot encoding cannot be applied since the cardinality of *geo_level_id_3* is more than 10000, and this would lead to a very high-dimensional problem. Therefore, to overcome this complication, we decided to apply Target Encoding [5] to these features. This encoding technique replaces every categorical value with some statistics computed on the posterior probability of the target given that specific value. In our case, the computation was a mean;
- *Scaling* - Some algorithms, such as Logistic Regression and Support Vector Machine, require normalization of the input data. Taking into account the presence of some outliers, we opted for the usage of the Robust Scaler. This scaler removes the median and scales the data according to the range between the 1st quartile (25th quantile) and the 3rd quartile (75th quantile). This scaler has been proven to be robust to outliers.

In order to reduce the verbosity of the code, we wrote an ad-hoc class called Preprocessor, which encapsulates the aforementioned preprocessing steps.

Models

The task is a multiclass classification one. We adopted, in order to solve the problem, some classic machine learning algorithms that have been also used to create a baseline. This baseline is an indicator of the impact of the preprocessing and the parameter tuning on the final score. These algorithms are:

- *Random Forest* - It uses multiple uncorrelated decision trees working individually to determine the class of a given input. Then the final prediction of the model is the majority vote of the decision trees. In order to decorrelate the trees and to reduce the variance, it relies on bagging (bootstrap aggregation).
- *Support Vector Machine* - This iterative algorithm look for the best hyperplane that separates two classes in the feature space. This hyperplane is built by maximizing the margin between classes. Moreover, using the kernel trick, it's possible to obtain non-linear decision boundaries.
- *Logistic Regression* - It is a statistical analysis method used to predict a binary outcome, based on prior observations of a data set. It estimates the probability of an event occurring, based on a given dataset of independent variables. Since the outcome is a probability, the dependent variable is bounded between 0 and 1. In logistic regression, a logit transformation is applied to the odds. That is the probability of success divided by the probability of failure.

The SVM algorithm, which is implemented natively for binary classification, can actually perform a multiclass classification using the One-vs-one schema. The same thing can be said for Logistic Regression, but the latter uses a One-vs-rest schema. The difference is that having C classes, in the One-vs-rest schema there are C decision boundaries, where each one separates one class from the union of the others, while in the One-vs-one schema, each decision boundary is trained on $\frac{C \cdot (C-1)}{2}$ binary problems that take into consideration only one pair of classes.

To handle the complexity of the problem, we performed deep studies on two modern boosting algorithms: LightGBM and CatBoost. The explanation of these algorithms will follow in the next sections.

Light GBM

Light Gradient-Boosting Method [6] is a boosting algorithm based on the classical Gradient Boosting Decision Tree, but it's much faster. The most time-consuming operation when building up a decision tree is the choice of the splitting feature. In every node, for each feature, it must estimate the information gain of all possible split points. The main idea is to reduce both the number of samples and the number of features. The first way to speed up the training is to discretize the numerical features using the histogram-based algorithm, which is used in many boosting algorithms. This algorithm buckets the values of continuous features into discrete bins, that are used during the search for the best split point. The number of samples is reduced using the Gradient-based One-Side Sampling (GOSS). In a boosting algorithm, we want to minimize the gradient of the loss, which is the contribution of the gradient of every instance. In other words, we want to correctly classify the misclassified points of the previous tree. If an instance has a small gradient, it means that it is almost well-trained. Therefore, no more work is required for it. The idea of GOSS is to keep all the instances having a large gradient (which are the misclassified points) and to sample a subset of those having a small gradient. In this way, the number of instances is reduced. The only problem is that this sample strategy would modify the data distribution. The algorithm solves that by adding a weight to the instances having a small gradient during the information gain estimation. The number of features is reduced using the Exclusive Feature Bundling (EFB). This algorithm works under the assumptions that high-dimensional data are usually very-sparse and that there are many mutually exclusive features. If these assumptions hold (and in our case they do), the mutually exclusive features can be bundled together. Moreover, the algorithm can allow small fractions of conflicts allowing us to get an even smaller number of feature bundles. Other important characteristics of LightGBM are its ability to handle categorical features [7] and that the decision trees are grown leaf-wise [8], which increases the accuracy of the models since the number of leaves is fixed by a hyperparameter.

CatBoost

The Catboost (Categorical Boosting) [9] is a gradient boosting library released by Yandex in 2017 that is designed for working with categorical data. Catboost handles the categorical features with a variant of the Target Statistics encoding called Ordered Target Statistics. This encoding method, inspired by online learning algorithms which get training examples sequentially in time, introduces artificial time. To do this, it performs a random permutation of the training samples and, for each sample, it uses all the available "history" to compute its Target Statistics. If only one permutation of the

dataset is used, for the initial few rows in the randomized data, the mean encoding will have high variance as it only saw a few points from the history. But, as it sees more data, the running average starts to get stable. This instability due to randomly chosen initial data points, makes the initial part of the dataset have weak estimates. Catboost handles this problem by using more than one random permutation. It trains several models simultaneously, each one is on its own permutation. The models on which Catboost relies are symmetric oblivious decision trees, which are trees that share the same symmetric structure, but the leaf values of these models are different. Before building the next tree, Catboost selects one of the models, which will be used to select the tree structure.

IV. Experiments, results, and discussions

In this section, we are going to illustrate the results of the grid search we implemented. The metric used to evaluate the models is the F_1 micro score, which is the multiclass variant of the F_1 score. For binary classification, the F_1 score is the harmonic mean between precision and recall, where precision is the number of samples correctly predicted as positive (True Positive) over the number of all samples predicted as positive (TP + False Positive), while recall is TP over the number of all samples that are really positive (TP + False Negative). For multiclass classification, the F_1 micro score is the harmonic of the precision and the recall among the classes.

First of all, in order to analyze the results, we ran three basic algorithms (Random Forest Classifier, Support Vector Classifier, and Logistic Regression) with their standard hyperparameters and without any relevant preprocessing, other then the one-hot encoding on categorical features. Among them, we choose the Random Forest as our baseline since it performed better than the others ($F_1 = 0.7104$).

A. Hyparameters tuning

Grid search is a very popular technique used to determine the hyperparameters of a certain model. In fact, for every combination of these parameters, we get a different model with different performances on the dataset. In grid search, the trial solutions are drawn from a regular grid in the model space. That means we choose discrete values of the parameters we're testing. These values are separated by certain intervals, like on a grid. The "Gridsearch" algorithm tests the combinations of these parameters and returns the one with the best performance. In our project, we performed a grid search for all of our models. The different parameters used are summarized in Table I.

In Table I, the best configuration of parameters for each model resulting from the grid search has been highlighted.

B. Training

Each model, as stated before, requires different preprocessing of data for the training. Unlike the Logistic Regression and SVM, non-scaled data won't affect the Random Forest performance. For this reason, we didn't scale its input. On the other hand, the Logistic Regression and the SVM required all the preprocessing steps listed above.

TABLE I: Grid search parameters

Model	Parameters	Values
Random Forest	*n_estimators*	[100, 500, **1000**]
	criterion	['gini', '**entropy**']
	min_samples_leaf	[1, **2**]
	min_samples_split	[2, **5**]
Logistic Regression	*penalty*	['**l2**, 'elasticnet"]
	tol	[**1e-4**, 1e-6]
	C	[0.5, **1.0**, 2.0, 5.0]
	solver	['newton-cg', 'lbfgs', '**saga**']
	max_iter	[1000, 5000, **10000**]
SVM	*C*	[0.1, **1**, 5]
	kernel	['**rbf**', 'linear']
	tol	[**1e-4**]
LightGBM	*num_leaves*	[20, **30**]
	learning_rate	[0.1, **0.05**, 0.01]
	n_estimators	[**5000**]
	feature_fraction	[**0.5**, 0.65, 0.8]
	min_child_weight	[**0.1**, 0.5]
	max_bin	[4096, **8192**]
CatBoost	*iterations*	[1000, 2000, **5000**, 7000]
	learning_rate	[0.02, **0.05**, 0.07]
	depth	[**9**, 10, 12]
	eval_metric	[**TotalF1**]
	l2_leaf_reg	[1, **2**, 5, 9]
	border_count	[**None**, 11, 15]

A different approach has been used for the boosting models. In fact, both LightGBM and Catboost are efficient in dealing with categorical features, and the input data does not require any scaling. However, while the Catboost performs its own effective encoding of categorical dimensions, in LightGBM [7], categorical features are converted to gradient statistics at each step of gradient boosting. Nevertheless, a better result was achieved when transforming the Geographical Ids columns with the Target Statistics. For this reason, only the TS encoder was applied to the LightGBM input.

C. Results

The results deriving from the submission on the Driven Data website are listed in Table II

TABLE II: Results

Model	F1 Score
Random Forest	0.7465
Logistic Regression	0.7247
SVM	0.7331
LightGBM	0.7497
CatBoost	0.7516

The best-performing model has proven to be the Catboost. The result was expected since the Catboost model is the state-of-the-art model between boosting algorithms for datasets that contain many categorical features, and this is due to its proficient auto-encoding of those features. Fig. 5 shows the progressive reduction of the training objective compared to the increase of the evaluation metric (F1-score). After the first 100 iterations, the two metrics start to assume a linear behavior. In general in the test set after about the 5000th iteration the performance of the model was stable. However, with higher iteration values the overfitting phenomenon started to manifest.

A way to have insight into the behavior of a Machine Learning model is to visualize the confusion matrix. This

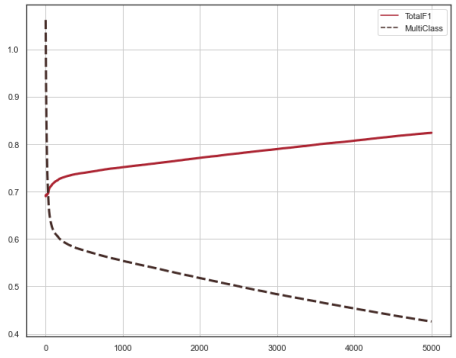


Fig. 5: Training metrics of the Catboost classifier

matrix highlights where the model is most likely to confuse two classes. Fig 6 shows the confusion matrix of the Catboost model. This matrix has been computed on the 20% of validation data from the 80/20 split of the training data.

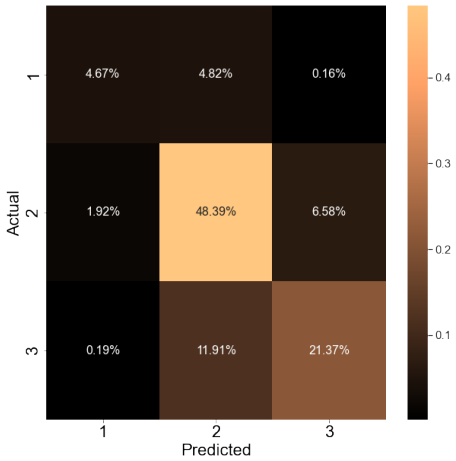


Fig. 6: Confusion Matrix of the Catboost

As expected, the model is more likely to classify an instance of medium damage (label 2) due to the imbalance in the original dataset. Most of the data with label 1 (low damage), instead, were misclassified into label 2, and the same happen for 1/3 of the data with label 3.

In order to weaken the misclassification, we decided to proceed with a variant of the ensemble method. With the classical ensemble strategy, the label is selected using a majority vote between multiple predictors. Instead, we decided to use the weighted sum of the estimated probability of model h_i. In mathematical terms we can say that, given the estimated probability p_i^c computed by the model h_i with weight w_i that the sample $x_k \in X$ has label c, the final prediction will be:

$$y_k = arg\max_c \sum_i w_i \cdot p_i^c$$

To compute this ensemble method, we relied on the three strongest learners, which are: the Random Forest, the LightGBM, and the Catboost. The weights associated with each model have been selected taking into account the behavior of each model in the misclassification process (a higher weight will have stronger decisive power in the ensemble). The selected weights are 0.8 for the Random Forest, 1.0 for the LightGBM, and 1.2 for the Catboost. With this configuration, we reached our highest f1 score of 0.7526, which is currently the 38th-best result in the competition, and placed us into the first 1% of the leaderboard.

V. Conclusions and Future Works

The Nepal earthquake was a tragic event that caused significant loss and suffering. However, the extensive dataset that was collected in its aftermath has provided valuable insights and information that can be used to improve the safety and security of buildings and communities in the future.

The Catboost model has proven to be the best algorithm for this classification task, where categorical features are the majority. Furthermore, our ensemble technique increased the predictive power of the models and improved the final f1 score.

As a future work, the database can be extended using some features available from the OpenData portal [10]. Other than the listed features, this database also contains some information about the location of the building in terms of latitude and longitude, along with some information about the social status of the inhabitants of the buildings.

VI. Contributions

Dario implemented the Random Forest and LightGBM, and also performed experiments on Ordinal Logistic Regression. Umberto implemented CatBoost, the ensemble of methods, and performed experiments on the Smote resampling technique of the imblearn library. Achraf implemented Logistic Regression and SVM. Data exploration, preprocessing and reporting were carried out by everyone.

References

[1] "Richter's predictor: Modeling earthquake damage." https://www.drivendata.org/competitions/57/nepal-earthquake/.
[2] "Nepal earthquakes: Devastation in maps and images." https://www.bbc.com/news/world-asia-32479909.
[3] A. Kumar, P. N. Hughes, V. Sarhosis, D. Toll, S. Wilkinson, R. Coningham, K. P. Acharya, K. Weise, A. Joshi, C. Davis, R. B. Kunwar, and P. N. Maskey, "Experimental, numerical and field study investigating a heritage structure collapse after the 2015 gorkha earthquake," 2020.
[4] D. P. Kroese, Z. I. Botev, T. Taimre, and R. Vaisman, *Data Science and Machine Learning*. 2020.
[5] D. Micci-Barreca, "A preprocessing scheme for high-cardinality categorical attributes in classification and prediction problems," *SIGKDD Explor. Newsl.*, vol. 3, p. 27–32, jul 2001.
[6] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "Lightgbm: A highly efficient gradient boosting decision tree,"
[7] "Lightgbm: Categorical feature support." https://lightgbm.readthedocs.io/en/latest/Advanced-Topics.html#categorical-feature-support.
[8] S. Haijian, "Best-first decision tree learning," 2007. The University of Waikato.
[9] L. Prokhorenkova, G. Gusev, A. Vorobev, A. V. Dorogush, and A. Gulin, "Catboost: unbiased boosting with categorical features," 2017.
[10] "2015 nepal earthquake - open data portal." https://eq2015.npc.gov.np/#/.