Predicting Earthquake Damage

Applying Machine Learning to predict damage from the 2015 Gorkha earthquake in Nepal

by

Alessio Ciullo

A report submitted for the Certificate of Advanced Studies
in Advanced Machine Learning at the University of Bern, Switzerland

April 2021

Abstract:

Predicting losses from natural hazard is crucial for both public and private agencies. This project uses four machine learning algorithms, i.e., logistic regression, random forest, gradient boosted trees, and neural networks, to predict earthquake damage from the 2015 Gorkha Nepali earthquake. Post-disaster data were collected by the Kathmandu Living Labs and the Central Bureau of Statistics surveying 762.106 buildings affected by the earthquake. For each building, 38 features were collected, and each building is categorized per damage grade from 1 to 5, i.e., from low to damage to complete destruction. The procedure is conducted using various resampled training set, as well as by reformulating the problem as a 3 classes classification problem, i.e., by merging the original 5 classes. This allows to compare performances with previous studies. It is found that tree-based models outperform the others for all classes irrespective of the resampled training set and whether 5 or 3 classes are used. Random forest proved the best performing algorithm with a micro-averaged F1 score of 0.596 and a micro-averaged AUC score above 0.85 for the 5 classes case, a micro-averaged F1 score of 0.722 and micro-averaged AUC score of almost 0.9 for the 3 classes case. This latter is slightly inferior but comparable to previous research, were a micro-averaged F1 score of 0.744 was achieved. All data and code can be accessed at https://bit.ly/3xFVjAG.

1. Introduction

One of the main challenges in modeling impact from extreme natural events, like floods, storms, or earthquakes, is the assessment of the vulnerability of the exposed objects, e.g., buildings, roads etc., to such events. Historically, this has been assessed using functions developed based on expert judgment. These functions are typically simple ones relating one feature (e.g., flood extent, wind velocity, ground motion etc.) to the registered damage extent. The large uncertainties in the definition of these functions are mainly due to the lack of data, which in turn is due to i) the intrinsic low likelihood of such events (they are extreme events by definition) and, even they happened, the ii) the difficulty of collecting data in the aftermath of these events. In the latest years, with the emerging of new data acquisition technologies, as well as the raising awareness regarding the importance of collecting such data, databases of damages from extreme natural events are increasing in number and size. As a consequence, the scientific community is taking advantage of this increase in data availability and it is applying new methods to improve the standard expert-based approach.

Merz et al., 2013 was among the first to introduce the use of machine-learning methods to analysis large damage data from floods. They demonstrated that tree-based methods outperformed classic methods. Using different data, Wagenaar (2017) came to the same conclusion and showed that tree-based methods also outperformed other non-standard methods like Bayesian Networks. However, Schröter et al. (2014) found that Bayesian Networks may be more powerful when transferring models in space, i.e., when using models trained on a case study to predict damage for another case study, for which no training data are available. Other authors focused on earthquake damages. Tesfamariam and Liu (2010) used machine learning algorithms and found that Support Vector Machine and k-nearest neighbors are among the best performers. Mangalathu et al. 2020 analyzed data from the 2014 South Napa earthquake, in the USA, and found that random forests algorithms could accurately predict 60 % of the buildings damage.

This project aims at predicting earthquake damage from the 2015 Gorkha Nepali earthquake using one of the largest post-disaster datasets ever collected in the context of natural hazard risk assessment. The data were collected by the Kathmandu Living Labs and the Central Bureau of Statistics surveying a large number of buildings affected by the earthquake. The goal is to analyze the available data, process them, and finally train logistic regression, random forest, gradient boosted decision trees and neural network algorithms to predict buildings damage grades. A very similar exercise was carried out by Adi et al. (2020), which analyzed a slightly different version of the same data used and which will be used as reference for discussion in the aftermath. Section 2 describes the dataset and how they are processed; Section 3 illustrates the method; Section 4 discuss results and Section 5 provides final conclusions.

2. Data description and pre-processing

The dataset can be freely downloaded from the 2015 Nepal Earthquake Open Data Nepal Portal (http://eq2015.npc.gov.np/). The portal provides several datasets regarding individuals, buildings, and households information. The goal of this project is to estimate damage levels based on information about buildings that can be acquired *before* the earthquake. The available info about buildings relate to the buildings' structure, damage assessment as well as ownership and use. The supplementary material provides an overview of all features contained in these three datasets and it highlights those selected for the analysis, i.e., those which can be acquired *ex-ante*. The selected dataset amounts to 762'106 instances and 38 features. The target variable, i.e., *damage_grade*, is a categorical variable with 5 categories, corresponding to five damage levels. Of the 38 features, 8 are categorical, 3 are textual (also a type of categorical variable), 22 are binary (0 or 1) and 5 are numeric.

2.1 Treating categorical features

The 8 categorical features are treated with the standard one-hot-encoding method, where, i.e., each feature is transformed into as many one-hot-encoded features as the number of categories in that feature. There is thus one one-hot-encoded feature per category. All one-hot-encoded features are zero except for the one relative to the category value in that instance, which is a one. One-hot-encoding has the advantage of transforming categorical features into a numeric format, which most machine-learning algorithms can more easily deal with, but it comes with the cost of enlarging the number of features.

When dealing with high-cardinality categorical features, which i.e., have many categories, one-hot-encoding can be problematic as the newly created features may be too many. This is the case for the 3 textual features, i.e., *district_id* (i.e., indicating in which of the 11 districts the building is located), *vdcmun_id* (i.e., indicating in which of the 110 municipalities the building is located), *ward_id* (i.e., indicating in which of the 945 ward numbers the building is located). For these features, impact (or target) encoding was used. Impact encoding was conducted via the *Category Encoders* package (https://contrib.scikit-learn.org/category_encoders/) and it was first introduce by Micci-Barreca (2001). It consists in assigning a numerical value to each category according to the target variable's expected value. Assuming a binary classification, a categorical vector

X and target variable y, the value of the i^{th} category x_i will be given by the probability $P_i = \mathrm{P}(y = 1 | X = x_i)$. Following Micci-Barreca (2001), this probability can be derived as:

$$P_i = \alpha(n_i)\frac{n_{iy}}{n_i} + (1 - \alpha(n_i))\frac{n_y}{n_{TR}}$$

where n_{TR} is the total number of instances, n_y is the number of instances such that $y = 1$, n_i is the number of instances where $X = x_i$, n_{yi} is the number of instances such that $y = 1$ and $X = x_i$, and α is a weighting factor that depends on n_i. In case of multi-class classification problems, like in the present work, the procedure is applied to the one-hot-encoded target variables. By so doing, at the end of the procedure, the 3 textual features were transformed into 15 numeric features (the 3 features times the 5 target classes). Finally, after transforming both low- and high- categorical features into numeric ones, and dropping the original features, the number of features increases from 39 to 80.

2.2 Resampling training data

The data were split into train and test data, corresponding respectively to 80 % (609’675) and 20 % (152’431) of the full dataset. Looking at the left panel of Figure 1, one sees that the original training set (*orig* for short in the aftermath) is imbalanced with class *Grade*_5 alone accounting for 35 % of the instances and *Grade_1* only accounting for 10 % of them. A typical way to deal with this problem is via resampling instances in order to better balance classes. Resampling can take the form of up-sampling, i.e., sample more of the minority classes, down-sampling, i.e., drop instances from the majority classes, or a combination of the two. In this work, both up-sampling and a combination of up-sampling and down-sampling have been applied, and in the latter case two different down-sampling techniques are explored. All resampling methods have been performed using the *imbalanced-learn* package (https://imbalanced-learn.org/stable/).

Up-sampling is performed via the *Synthetic Minority Over-sampling Technique for Nominal and Continuous* (SMOTENC) technique, which adapt the *Synthetic Minority Over-sampling Technique* (SMOTE) introduced by Chawla et al. (2002) to deal with datasets that contains both numerical and categorical features (even if the categorical features where transformed into numeric, they are still discrete features). Given an instance in the minority class, SMOTE creates new samples by considering the k nearest neighbours in the feature space to this instance. Then it takes the vector between one of those k neighbours and the instance itself and finally multiplies this vector by a random number between 0 and 1. The result represents a new, synthetic instance. As up-sampling can generate noisy samples during the interpolation process, i.e., points very close to each other which belong to different classes, cleaning down-sampling techniques can be applied.

The applied down-sampling techniques are Tomek links and Edited Nearest Neighbors (ENN). Tomek links were introduced in Tomek (1976) and are defined as pairs of nearest-neighbor instances which belong to different classes. Once a Tomek link is identified, either both or only one instance (e.g., the one belonging to the majority class) is dropped. ENN was introduced by Wilson (1972) and it removes instances which do not agree (i.e., belong to the same class) with either the majority or all their nearest-neighbors. In this application, the majority rule has been adopted, which is obviously the more conservative of the two rules, i.e., it drops less instances.

The left panel of Figure 1 shows the prevalence of classes across the different datasets. Up-sampling with SMOTENC (*smotenc* for short in the aftermath) generates an equal number of classes with a total number of training instances of 1’104’100, cleaning with Tomek links (*smotenc_tmk* for short in the aftermath) reduces the number of training instances to 897’616 but overall maintains an equal distribution of classes, cleaning with ENN (*smotenc_enn* for short in the aftermath) reduces the number of training instances to 526’650 (lower than instances in *orig*) and also provides an imbalanced set, this time with the majority class being *Grade_1*.

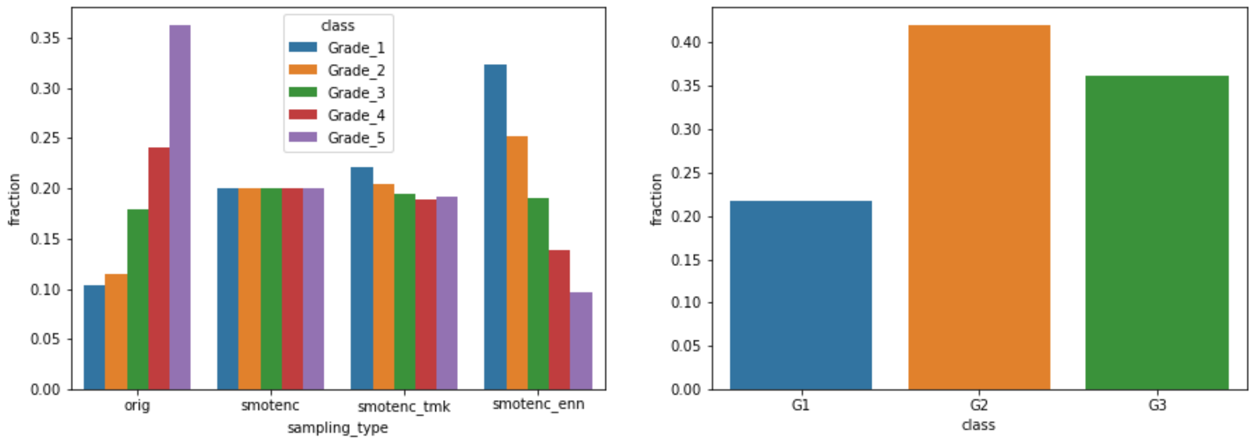

Figure 1 Class prevalence for the 5 classes case and different resampled training sets (left) and the 3 classes case (right).

2.3 Reduction of the number of classes

Given the problem at hand, not all wrong predictions have the same impact. Trivially, predicting e.g., a *Grade_2* damage when it is instead a *Grade_5* does not have the same negative consequences of predicting a *Grade_5* damage when it is instead a *Grade_2* damage. To avoid such disparity in the impact of misclassification, the 5 classes have been grouped into three categories, by merging the first class with the second, and the third with the fourth. This is not the most elegant way to approach the problem, but it certainly reduces the effect of wrong misclassifications as the three classes are somewhat “closer” to each other. The resulting classes distribution is shown in the right panel of Figure 1. No resampling has been carried out.

3. Method

The multi-class classification is performed using four different machine learning algorithms: logistic regression (*log* for short in the aftermath), random forests (*rf* for short in the aftermath), gradient tree boosting (*xgb* for short in the aftermath) and neural networks (*nn* for short in the aftermath). For the first three algorithms, tuning was carried out via a grid-search, while for neural networks various architectures and training parameters were manually tested. The performance of each algorithm is evaluated by the micro F1-score, the Receiver Operating Characteristic (ROC) curve and the Area Under the ROC Curve (AUC) measure evaluated for each class and for the micro-average across classes.

3.1 Parameters tuning

Table 1 summarizes the adopted setting for the grid search for *log*, *rf* and *xgb*. The reported parameters names correspond to variables in the *scikit-learn* package (https://scikit-learn.org/stable) for *log* and *rf* and the *XGBoost* package (https://xgboost.readthedocs.io/en/latest) for *xgb*. For the latter, the grid search was conducted with an early stopping procedure with a performance check based on the multi-class error rate *merror* carried out every 10 rounds.

Table 1 Grid search settings and results for the tuning of log, rf and xgb.

Algorithm	**Parameter**	**Grid search values**	**Best value**
log	*C*	10^{-3},1,10^{3}	1
log	*penalty*	*l1, l2, elasticnet*	*l2*
rf	*n_estimators*	50, 100, 200	200
rf	*max_depth*	5, 10, 20	20
rf	*max_features*	5, 10, 12	12
xgb	*n_estimators*	50, 100, 200	200
xgb	*max_depth*	5, 10, 20	10
xgb	*learning_rate*	0.08, 0.15, 0.35	0.15

For the *nn* algorithm, three model architectures are explored. The first architecture has 6 layers, with 200, 100, 50, 20, 10, 5 neurons per layer. The second architecture adds one layer of 500 neurons at the beginning of the first architecture. The third architecture adds one layer of 1000 neurons at the beginning of the second architecture. In all architectures, all but the last neurons used a *relu* activation function, while the last neuron used a *softmax* function. The loss was computed with *sparse categorical entropy*. Table 2 shows the various training settings tested, with parameters names reported as in the *TensorFlow* package (https://www.tensorflow.org/learn). The best setting is number 8.

Table 2 Training settings for nn.

Setting	***lr***	***epochs***	***batch_size***	***Architecture***
1	0.001	100	100	first
2	0.0001	100	100	first
3	0.001	100	100	second
4	0.001	60	60	second
5	0.01	40	60	second
6	0.05	40	60	second
7	0.1	20	60	second
8	**0.05**	**15**	**60**	**second**
9	0.001	100	100	third

3.2 Performance metrics

Micro-averaged F1 scores is defined as the harmonic mean of the micro-averaged precision and micro-averaged recall. These latter two are the precision and recall calculated using the *overall* true positives, false positives and false negatives, i.e., by summing them up across classes. Micro-averaging is preferred in case of unbalanced datasets and it differs from standard (macro) averaging as in the latter little weight is given to misclassified minority classes.

ROC curves and associated AUC are calculated for each class individually, i.e., by plotting the class' own false positive rates vs true positive rates, as well as by using the micro-averaged false positive rates vs true positive rates, which provide an overall performance of the classifier across classes.

4. Results

The micro-averaged F1-score results for all algorithms and data sets are shown in Table 3. Each column in the table shows the performance of one of the four introduced algorithms, plus a dummy classifier which classifies instances randomly. This latter is added to provide an idea on how much each algorithm improves classification. Rows show results across different datasets configurations, i.e., the first four showing different resampled training sets while the last show results for the 3 classes case.

The tree-based algorithms, i.e., *rf* and *xgb*, tend to have very similar performances and are clearly superior to the other for all cases. For the 5 classes case, the highest micro-averaged F1-score is achieved by *rf* with *orig*.

Table 3

classifier/data set	*dummy*	*log*	*nn*	*rf*	*xgb*
orig	0,199142	0,549492	0,571372	**0,596724**	0,592426
smotenc	0,199588	0,535314	0,575	0,586948	**0,590576**
smotenc_tmk	0,198092	0,53351	0,568925	0,584724	**0,58969**
smotenc_enn	0,200894	0,513742	0,544151	0,555029	**0,55516**
3Classes	0,335654	0,685708	0,70505	**0,722449**	0,720068

This is somewhat surprising as 1) *xgb* is expected to perform better than *rf* and 2) there seems to be no value in resampling. For the 3 classes case, *rf* is also the best classifier. Overall, when compared with a dummy random classifier, *rf* can increase the micro-averaged F1-score by about 300% in the 5 classes case and of more than 200% in the 3 classes case.

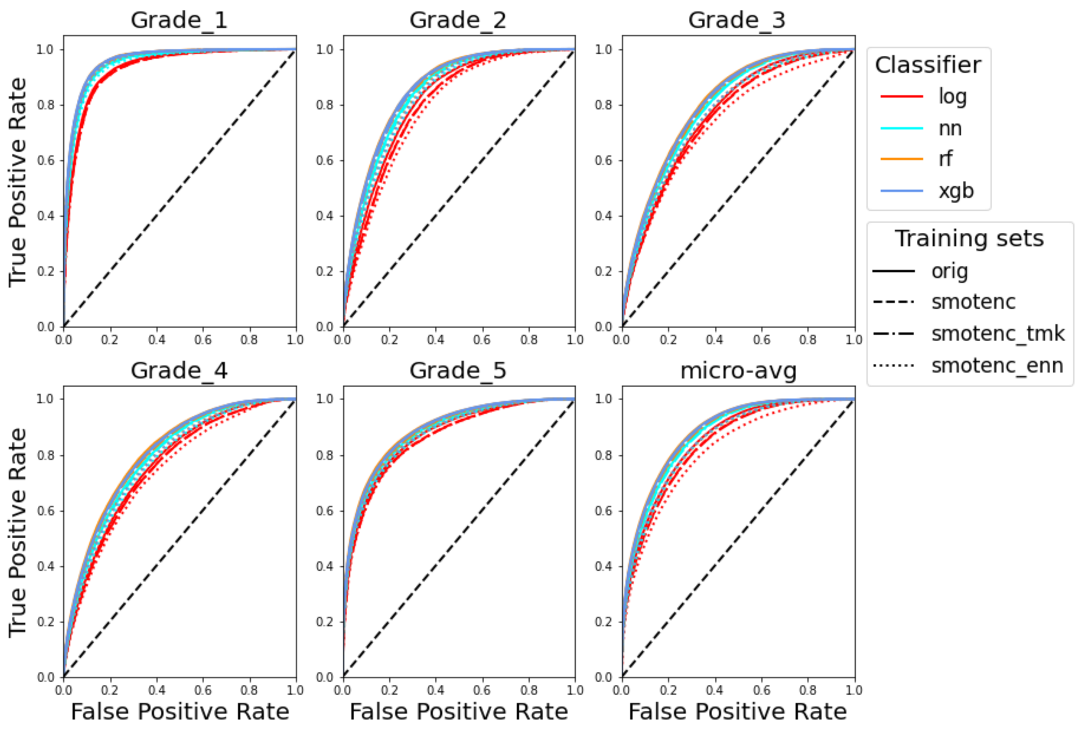

Figure 2 Classes and micro-averaged ROC curves across algorithms and resampled training data for the 5 classes case

Figure 2 and 3 respectively show the ROC curves and AUC for each class and the micro-averaged values for the 5 classes case. From the ROC curves we can see that all classifiers do a fairly good job at classifying class *Grade_1*. Surprisingly, this class is the minority class within the original data as shown in Figure 1. This partly explains why resampling did not help much with the micro-averaged F1 score, as the resampled class could already be well classified in the original data, and in a better way than classes with more instances. We also see that *xgb* and *rf* are the best classifiers for all classes regardless of the training dataset, and that log is the worst classifier.

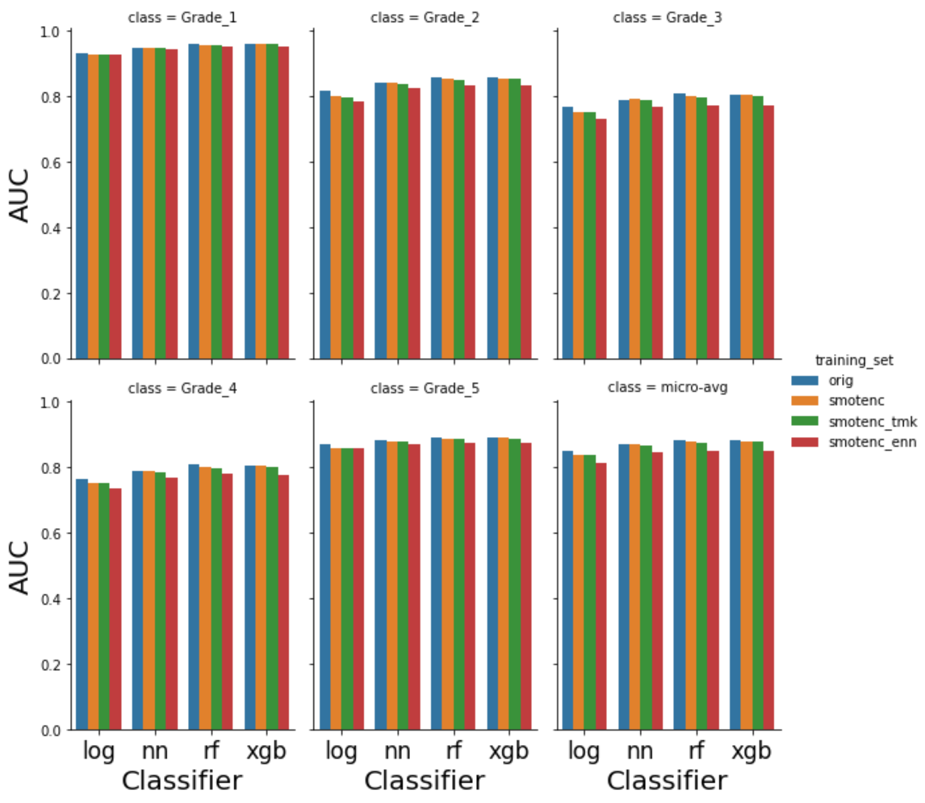

Figure 3 Classes and micro-averaged AUC score across algorithms and resampled training data for the 5 classes case

Grade_3 and *Grade_4* seem to be the two classes least well classified, while *Grade_2* seems to have the largest variability of ROC curves depending on the selected classifier and training set. Similar conclusions can be drawn by looking at the AUC. *Grade_1* has a very high AUC (over 0.9) and *Grade_3* and *Grade_4* have the lowest AUC (just below or above 0.8 depending on the classifier and training set). Overall, *xgb* and *rf* with *orig* allow reaching a micro-averaged AUC above 0.85.

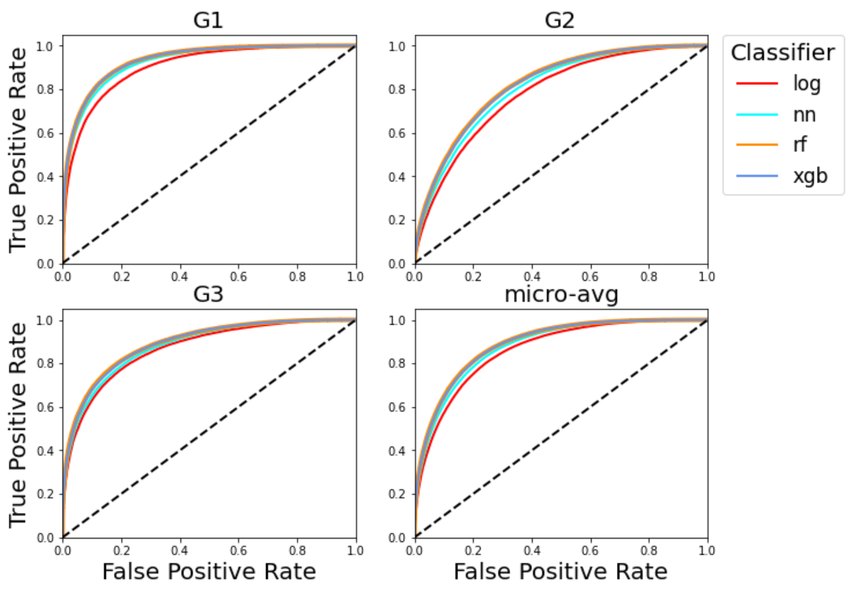

Figure 4 Classes and micro-averaged ROC curves across algorithms for the 3 classes case

Figure 4 and 5 respectively show the ROC curves and AUC score for each class and the micro-averaged values for the 3 classes case. The tree-ensemble models are the best classifiers for all classes. Class *G1* is the best classified class again from the two tree-based methods, but *nn* also seems to have almost equally good performances. The least well classified class is *G2*, and this is not surprising as it groups *Grade_3* and *Grade_4*, i.e., the two worst classified classes in the 5 classes case. The AUC score is over 0.9 for the *G1* class for almost all classifiers, and around 0.8 for class *G2*. Overall, tree-based methods can reach a micro-averaged AUC of almost 0.9.

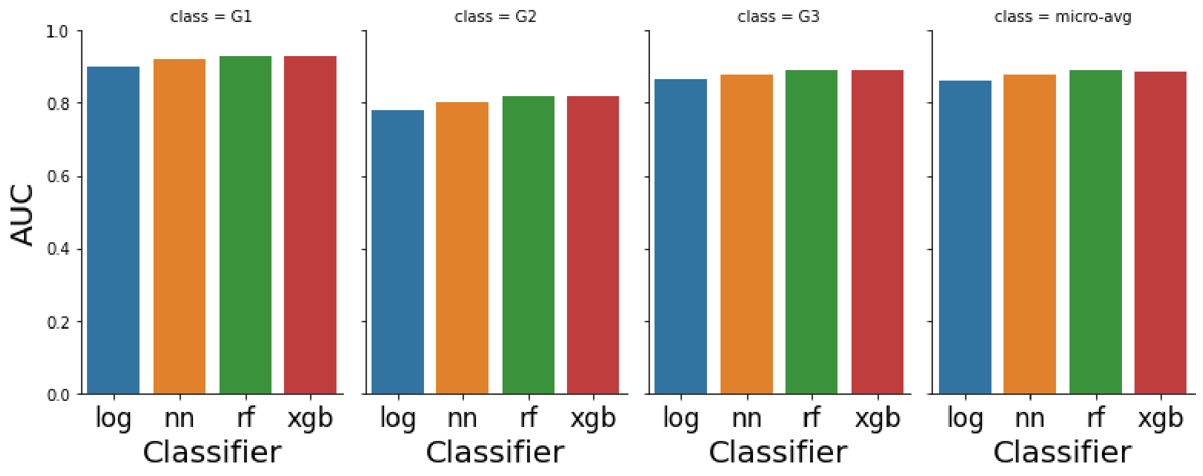

Figure 5 Classes and micro-averaged AUC score for the 3 classes case

5. Conclusions

This project aimed at classifying classes of damage grade from data collected after the 2015 Gorkha earthquake in Nepal. To achieve it, four machine learning algorithms were used, i.e., logistic regression, random forest, gradient boosted trees and neural networks. Since the data are imbalanced, various resampling techniques were used, using SMOTENC for up-sampling and Tomek Links and Extend Nearest Neighbor for down-sampling. The algorithms are fine tuned using grid search over the original training data, and then trained across all training datasets. The original problem is a five classes classification problem, but the procedure was also repeated for a three classes classification version of the original problem.

It is found that tree-based models outperform the others irrespective of the resampled training set used and the class. Random forest proved the best performing algorithm. For the original 5 classes formulation, random forest achieves a micro-averaged F1 score of 0.596 and a micro-averaged AUC score above 0.85. It is also found that, notwithstanding the imbalanced data, the resampled training set do not increase performance much. This is mostly due to the fact that class *Grade_1*, the minority class, is actually already classified very well in the original set.

For the 3 classes representation, a problem closer to what addressed by Adi et al. (2020) (although the three classes are grouped differently), random forest achieves a micro-averaged F1 score of 0.722 and micro-averaged AUC score of almost 0.9. Adi et al. (2020) found gradient boosted tree to be the best performing algorithm, with a micro-averaged F1 score of 0.744. This score is slightly higher than what achieved here. However, it is worth stressing that the grid-based parameter search in this project has been performed using the 5 classes formulation. Performing it on the 3 classes formulation may increase performances further.

References

Adi, Sourav Pandurang, Vivek Bettadapura Adishesha, Keshav Vaidyanathan Bharadwaj, and Abhinav Narayan. 2020. 'Earthquake Damage Prediction Using Random Forest and Gradient Boosting Classifier'. *American Journal of Biological and Environmental Statistics* 6 (3): 58. https://doi.org/10.11648/j.ajbes.20200603.14.

Chawla, N. V., K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. 2002. 'SMOTE: Synthetic Minority Over-Sampling Technique'. *Journal of Artificial Intelligence Research* 16 (June): 321–57. https://doi.org/10.1613/jair.953.

Mangalathu, Sujith, Han Sun, Chukwuebuka C. Nweke, Zhengxiang Yi, and Henry V. Burton. 2020. 'Classifying Earthquake Damage to Buildings Using Machine Learning'. *Earthquake Spectra* 36 (1): 183–208. https://doi.org/10.1177/8755293019878137.

Merz, B., H. Kreibich, and U. Lall. 2013. 'Multi-Variate Flood Damage Assessment: A Tree-Based Data-Mining Approach'. *Natural Hazards and Earth System Sciences* 13 (1): 53–64. https://doi.org/10.5194/nhess-13-53-2013.

Micci-Barreca, Daniele. 2001. 'A Preprocessing Scheme for High-Cardinality Categorical Attributes in Classification and Prediction Problems'. *ACM SIGKDD Explorations Newsletter* 3 (1): 27–32. https://doi.org/10.1145/507533.507538.

Schröter, Kai, Heidi Kreibich, Kristin Vogel, Carsten Riggelsen, Frank Scherbaum, and Bruno Merz. 2014. 'How Useful Are Complex Flood Damage Models?' *Water Resources Research* 50 (4): 3378–95. https://doi.org/10.1002/2013WR014396.

Tesfamariam, Solomon, and Zheng Liu. 2010. 'Earthquake Induced Damage Classification for Reinforced Concrete Buildings'. *Structural Safety* 32 (2): 154–64. https://doi.org/10.1016/j.strusafe.2009.10.002.

Tomek, Ivan. 1976. 'Two Modifications of CNN'. *IEEE Transactions on Systems, Man, and Cybernetics* SMC-6 (11): 769–72. https://doi.org/10.1109/TSMC.1976.4309452.

Vogel, K, C Riggelsen, F Scherbaum, K Schröter, H Kreibich, and B Merz. 2014. 'Challenges for Bayesian Network Learning in a Flood Damage Assessment Application'. In *Safety, Reliability, Risk and Life-Cycle Performance of Structures and Infrastructures*, edited by George Deodatis, Bruce Ellingwood, and Dan Frangopol, 3123–30. CRC Press. https://doi.org/10.1201/b16387-452.

Vogel, Kristin, Carsten Riggelsen, Bruno Merz, Heidi Kreibich, and Frank Scherbaum. n.d. 'Flood Damage and Influencing Factors: A Bayesian Network Perspective', 9.

Wagenaar, Dennis, Jurjen de Jong, and Laurens M. Bouwer. 2017. ‘Multi-Variable Flood Damage Modelling with Limited Data Using Supervised Learning Approaches’. *Natural Hazards and Earth System Sciences* 17 (9): 1683–96. https://doi.org/10.5194/nhess-17-1683-2017.

Wilson, Dennis L. 1972. ‘Asymptotic Properties of Nearest Neighbor Rules Using Edited Data’. *IEEE Transactions on Systems, Man, and Cybernetics* SMC-2 (3): 408–21. https://doi.org/10.1109/TSMC.1972.4309137.

Xie, Yazhou, Majid Ebad Sichani, Jamie E Padgett, and Reginald DesRoches. 2020. ‘The Promise of Implementing Machine Learning in Earthquake Engineering: A State-of-the-Art Review’. *Earthquake Spectra* 36 (4): 1769–1801. https://doi.org/10.1177/8755293020919419.