Using machine learning to predict building damage in the event of an earthquake

Anil Gautam

08/03/2022

Contents

0.1 Introduction

This project is part of an intermediate-level practice competition run by drivendata (Source https://www.drivendata.org/competitions/57/nepal-earthquake/). I picked this project because I wanted to be motivated at a personal level as well, I grew up in Kathmandu-Nepal. I was looking for anything related to Nepal and or Kathmandu on publicly available platforms. And I came across this project on drivendata competition page.

In April 2015, a large earthquake hit Nepal that killed thousands of people and cause widespread damage. Millions of people were made homeless including my friends and relatives. It is estimated Nepal incurred economic loss of 10 billion USD. It took years for the recovery effort, during this rebuild and recovery process, the Nepalese government with the help of private organisation Kathmandu Living Labs and the Central Bureau of Statistics, has generated one of the largest datasets ever collected. The dataset contains information on the extent of building damage, household conditions, building structure, geographical location information, land use, and other socio-economic-demographic statistics.

It can be useful to recognize structures and areas that are more susceptible to earthquake vibrations, this can be important for both building earthquake resistant structures and also in prioritizing post-disaster response such as focusing on retrofitting buildings that are most dangerous. This project explores the use machine learning algorithms to in damage prediction with the hope, I will be used to complement other in-ground tools such as rapid visual assessment.

0.2 Study Objective

This project aims to build predictive model for classifying ordinal variable damage_grade, which represents the extent of damage to a building, the dataset classifies damage into 3 categories:

1 represents low damage 2 represents a medium amount of damage 3 represents almost complete destruction

0.3 Literature review/ Previous work / Originality

There is not a lot of work done on this dataset. Google scholar and Github website were searched for related works, only one or two projects were found related to this dataset using R. My work builds on these previous works however no previous work has been directly copied.

A number of R packages have been used in this project and they are cited at the end. Some of the methods used for data wrangling, visualization and machine learning comes from the coursework (Irizarry, R. A. (2019). This online coursebook is freely available and I would like to acknowledge the author for his goodwill. All other work that were used in this project has been cited at the end.

0.4 Dataset and Features

Three data sets were provided by drivendata org for the challenge, (Source: https://www.drivendata.org/competitions/57/nepal-earthquake/), these data sets were uploaded to github for subsequent easier access. Of the three data sets only train and labels data were used in this project. Test data provided by the organisation was solely for competition submission. Two data sets train (building_id and their features) and label (building_id and damage_grade) were combined into one dataset for this project.

There are 39 columns in this dataset, where the building_id column is a unique and random identifier. The remaining 38 features are related to: geographical location (at three different levels), structural information such as age, area and height, number of floors, foundation type, roof type, structural material (mud, brick, timber, cement mortar), it's ownership and usage such as number of families, building usage (school, hotel, rental etcetra). The categorical variables were random lowercase ASCII characters, and appear to have no relation to what they were. The feature details are given below:

```
## 'data.frame':    260601 obs. of  40 variables:
##  $ building_id                           : int  802906 28830 94947 590882 201944 333020 728451 475515
##  $ geo_level_1_id                        : int  6 8 21 22 11 8 9 20 0 26 ...
##  $ geo_level_2_id                        : int  487 900 363 418 131 558 475 323 757 886 ...
##  $ geo_level_3_id                        : int  12198 2812 8973 10694 1488 6089 12066 12236 7219 994
##  $ count_floors_pre_eq                   : int  2 2 2 2 3 2 2 2 2 1 ...
##  $ age                                   : int  30 10 10 10 30 10 25 0 15 0 ...
##  $ area_percentage                       : int  6 8 5 6 8 9 3 8 8 13 ...
##  $ height_percentage                     : int  5 7 5 5 9 5 4 6 6 4 ...
##  $ land_surface_condition                : Factor w/ 3 levels "n","o","t": 3 2 3 3 3 3 1 3 3 3 ...
##  $ foundation_type                       : Factor w/ 5 levels "h","i","r","u",..: 3 3 3 3 3 3 3 5 3 1
##  $ roof_type                             : Factor w/ 3 levels "n","q","x": 1 1 1 1 1 1 1 2 2 1 ...
##  $ ground_floor_type                     : Factor w/ 5 levels "f","m","v","x",..: 1 4 1 1 1 1 4 3 1 3
##  $ other_floor_type                      : Factor w/ 4 levels "j","q","s","x": 2 2 4 4 4 2 2 4 2 1 .
##  $ position                              : Factor w/ 4 levels "j","o","s","t": 4 3 4 3 3 3 3 3 3 3 .
##  $ plan_configuration                    : Factor w/ 10 levels "a","c","d","f",..: 3 3 3 3 3 3 3 10 3
##  $ has_superstructure_adobe_mud          : int  1 0 0 0 1 0 0 0 0 0 ...
##  $ has_superstructure_mud_mortar_stone   : int  1 1 1 1 0 1 1 0 1 0 ...
##  $ has_superstructure_stone_flag         : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ has_superstructure_cement_mortar_stone: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ has_superstructure_mud_mortar_brick   : int  0 0 0 0 0 0 0 0 0 0 ...
```

```
##  $ has_superstructure_cement_mortar_brick: int  0 0 0 0 0 0 0 1 0 1 ...
##  $ has_superstructure_timber             : int  0 0 0 1 0 0 0 1 1 0 ...
##  $ has_superstructure_bamboo             : int  0 0 0 1 0 0 0 0 0 0 ...
##  $ has_superstructure_rc_non_engineered  : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ has_superstructure_rc_engineered      : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ has_superstructure_other              : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ legal_ownership_status                : Factor w/ 4 levels "a","r","v","w": 3 3 3 3 3 3 3 3 3 3 .
##  $ count_families                        : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ has_secondary_use                     : int  0 0 0 0 0 1 0 0 0 0 ...
##  $ has_secondary_use_agriculture         : int  0 0 0 0 0 1 0 0 0 0 ...
##  $ has_secondary_use_hotel               : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ has_secondary_use_rental              : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ has_secondary_use_institution         : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ has_secondary_use_school              : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ has_secondary_use_industry            : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ has_secondary_use_health_post         : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ has_secondary_use_gov_office          : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ has_secondary_use_use_police          : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ has_secondary_use_other               : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ damage_grade                          : Factor w/ 3 levels "1","2","3": 3 2 3 2 3 2 3 1 2 1 ...
```

The data set was evaluated to see how many unique values were present in each feature. Figure (1) shows most variables were appear either binary, categorical or discreet numerical.

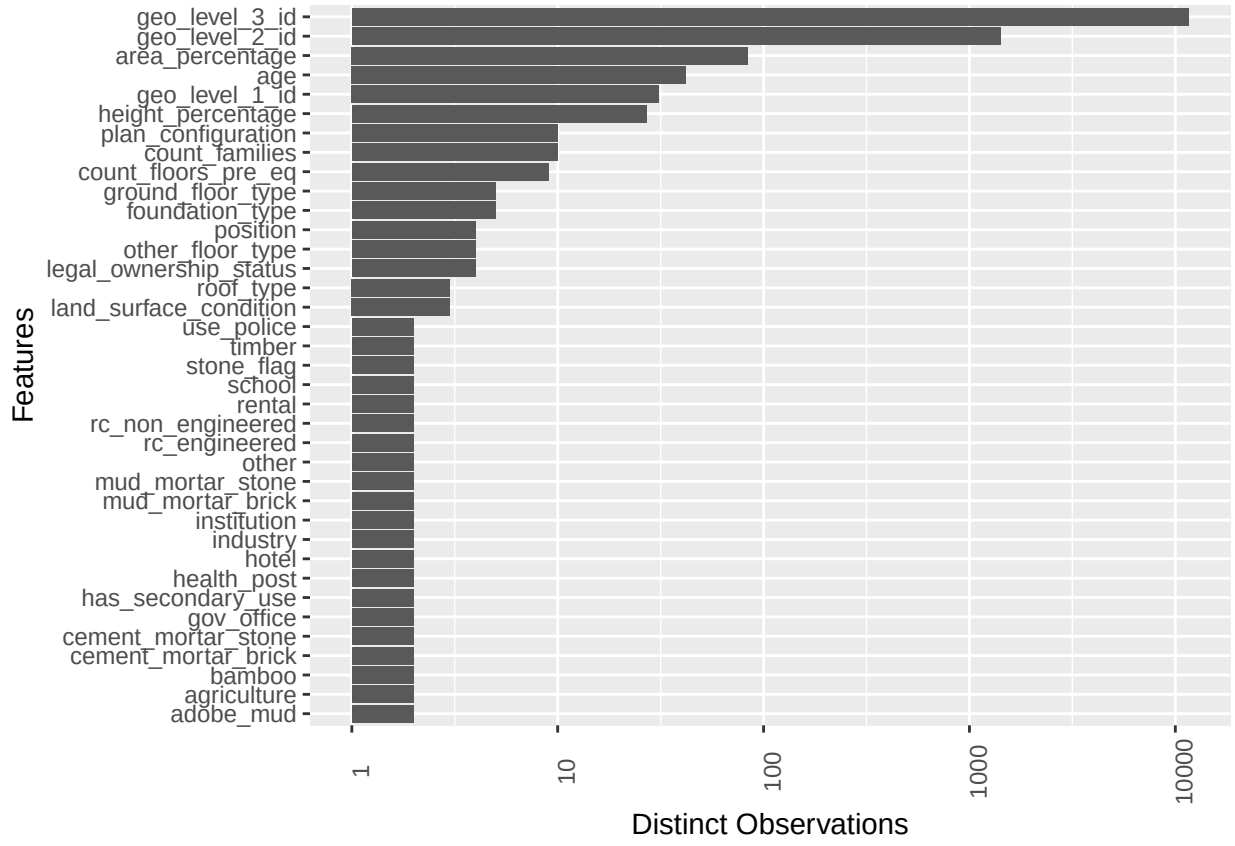


Figure 1: How many distinct observation per feature

/new page

0.5 Data Exploration

Simple check showed (sum(is.na(dfqk)==TRUE)) there were no missing values in the data set.

0.5.1 Predict Variable (Damage Grade)

The distribution of damage_grade, Figure (2) shows the prevalence of medium level damage and rarity of low damage.This kind of class imbalance can affect how well the algorithms perform for rare classes. There are a number of ways of addressing class imbalance by using under or over sampling. However none were used in this exercise. Attempt was made to address this within the model, this will be discussed as part of the model.

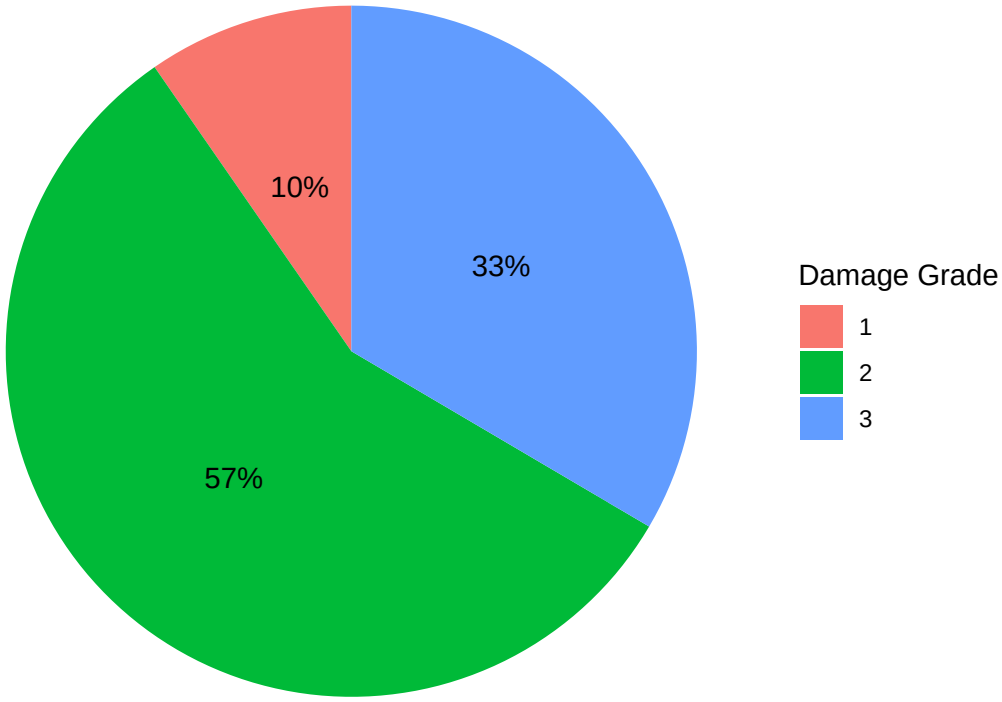


Figure 2: Class imbalance

0.5.2 Categorical Variables

Figure (3) shows an unequal proportion of damage_grade for most of these categories. Some categories appear to have a lot more influence on the type of damage, for example there is a lot less number of heavily damaged building for foundation_type of “i”, ground floor type “v” and roof type “x”.

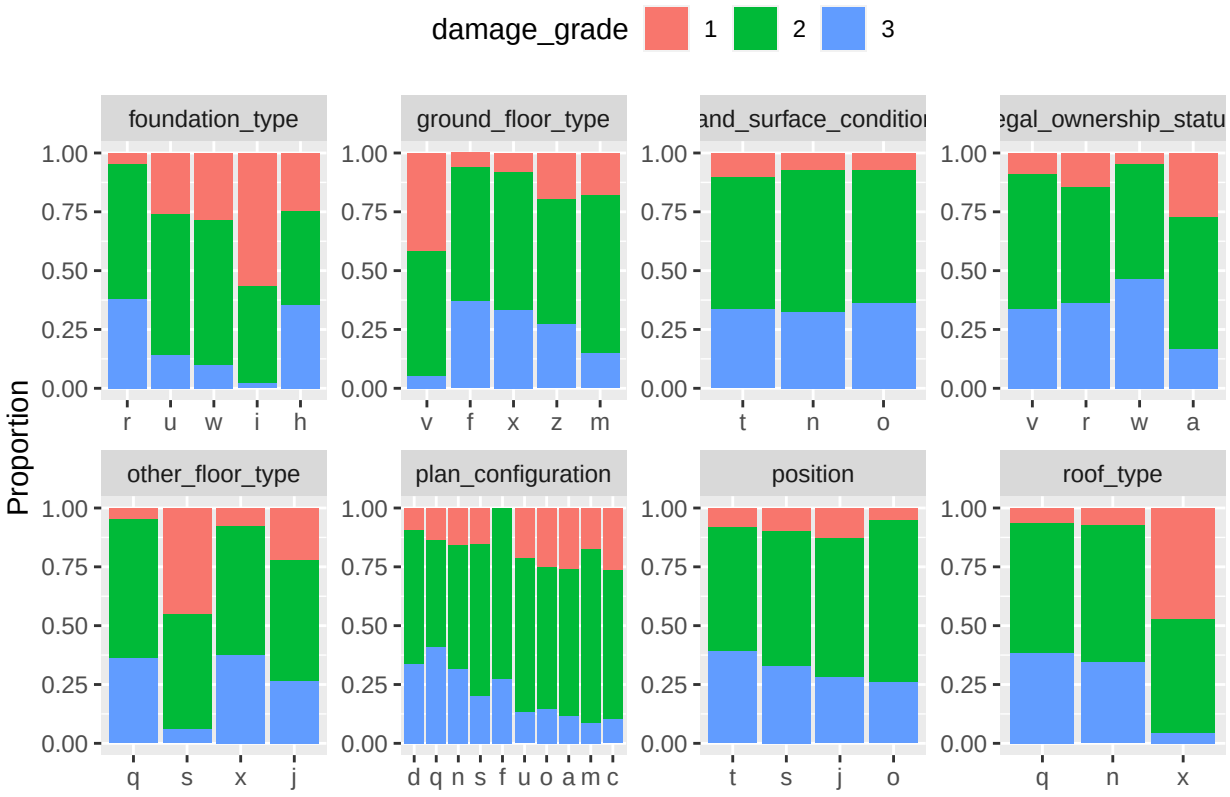


Figure 3: Categorical variable and their effect

0.5.3 Building Material

Figure (4) shows how building material could be an important variable as well in predicting building damage. In these figures, 1 represents the presence and 0 represents their absence. For example, building with reinforced concrete material showed lot less class 3 damage.

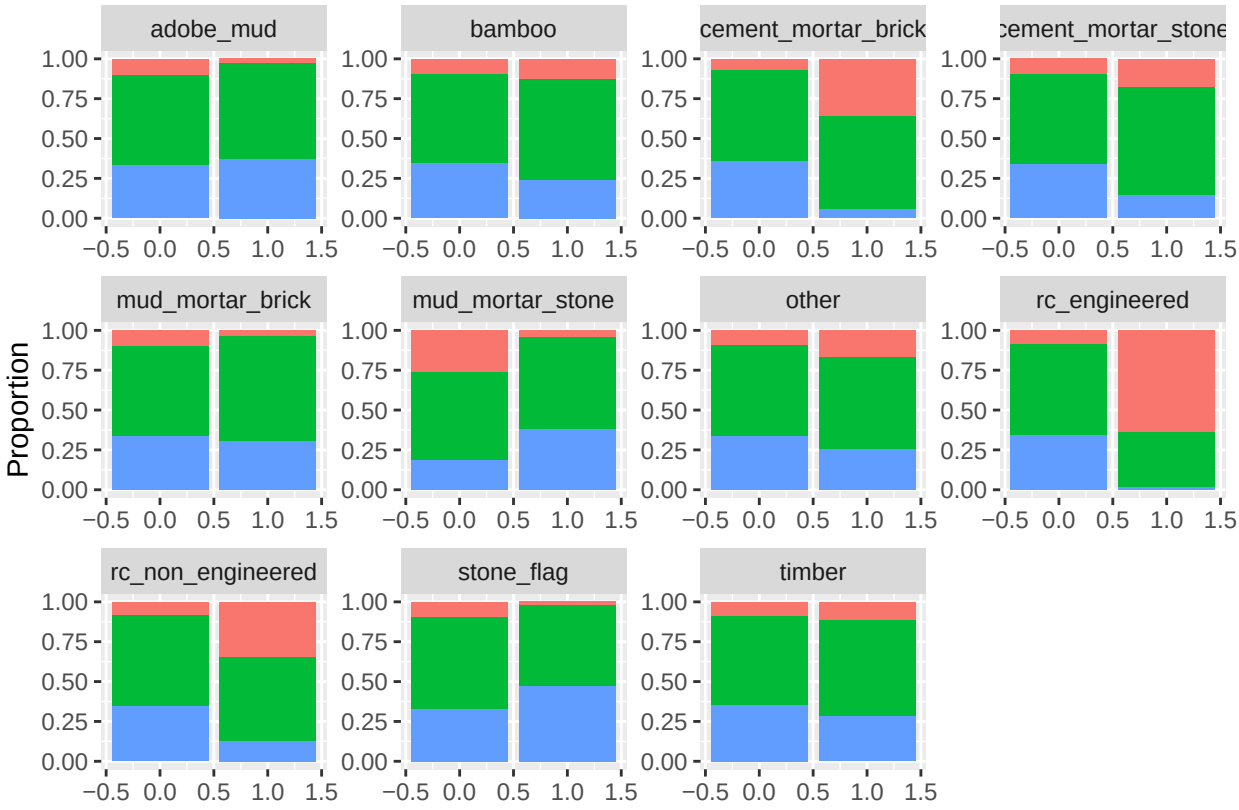


Figure 4: Building material and their effect

0.5.4 Building Usage

Figure (5) shows secondary usage has some effect on building damage, generally a protecting one particularly for buildings that were also used as government office, institutes and rental.

0.5.5 Location and dimension (Numerical)

Features like age, geographical location, area and height were presented as numerical variable. These numerical variables were first checked for interdependence. The correlation plot (6) showed strong correlation between number of floors and height, this could be important if we wanted to remove one of the variables for subsequent machine learning algorithm.

Figure (7) show all numerical variables show a trend with regard to damage grade. For example, with age there is a reduction in damage grade 1 suggesting as building ages it's more likely to suffer heavier damage. Unlike age, height seems to have protecting effect at either ends (very short or very tall buildings) but has negative effect in the middle. Location at level 1 and 2 seem to be more discriminating compared to level 3.

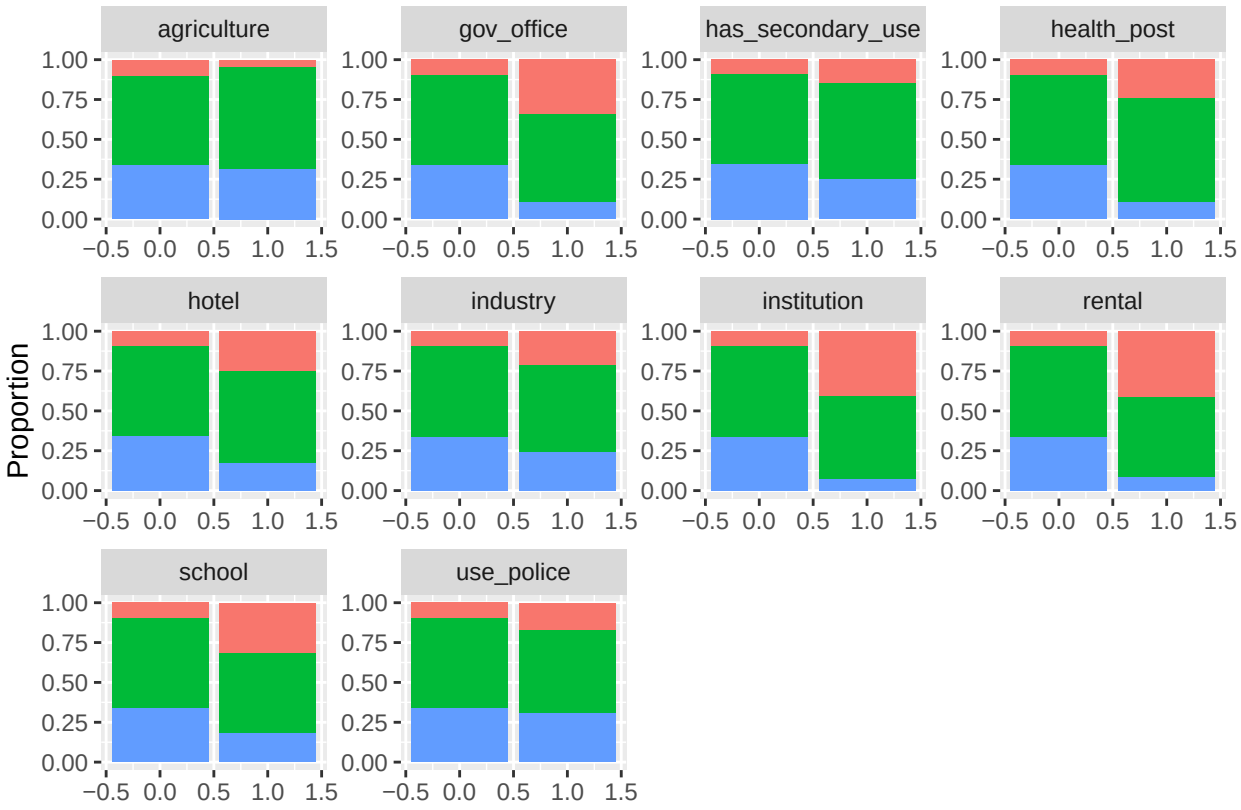


Figure 5: Building secondary usage and their effect

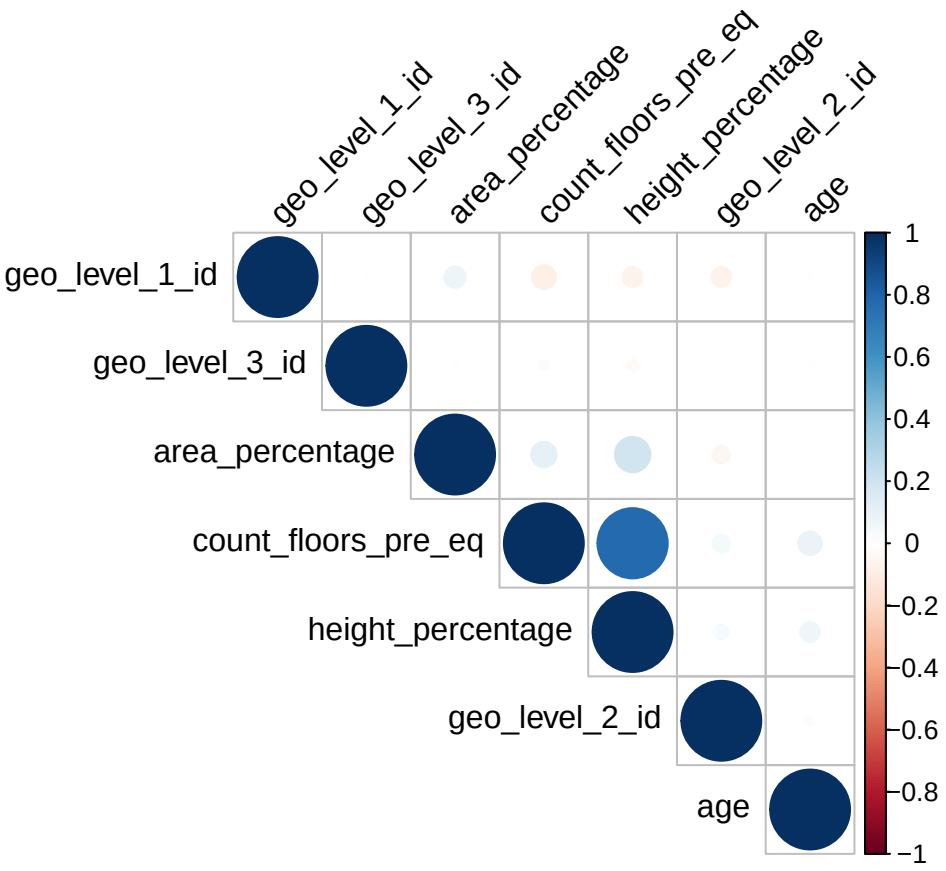


Figure 6: Corelation plot

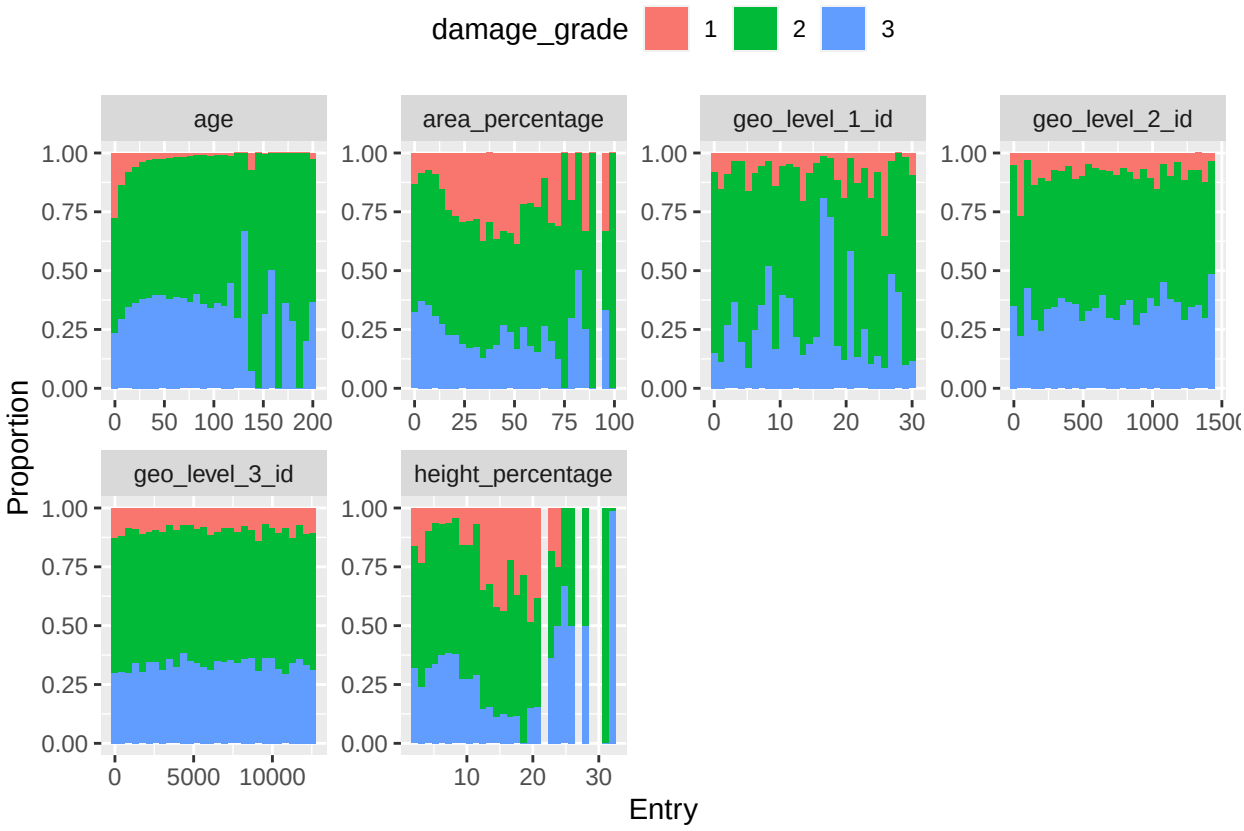


Figure 7: Location and dimension: Numerical Variables

In addition, there is a number of buildings (>1000) with age more than 900 years. I suspect this is erroneous data, although there are few old buildings and palaces they can't all be of the same age. Also it would have been very difficult to accurately date buildings in Nepal to that level of precision due to the lack of historical records before 1000 BC. In reality these data should be removed, however I see similar aged buildings are also present in the test data set and therefore for competition sake, these will be included.

0.6 Machine Learning Methods

Based on the variety of data, lack of normally distributed variables and a mix of categories, binaries and numerical variables, decision tree based algorithms were conisdered appropriate for this project. This is supported by previous works on this by Harirchian et al. (2022), Zhang et al. (2018) and Liang, Y. (2021).

0.6.1 Data Preparation

```
##    land_surface_condition.n land_surface_condition.o land_surface_condition.t
## 1                         0                        0                        1
## 2                         0                        1                        0
## 3                         0                        0                        1
## 4                         0                        0                        1
## 5                         0                        0                        1
## 6                         0                        0                        1
## 7                         1                        0                        0
## 8                         0                        0                        1
## 9                         0                        0                        1
## 10                        0                        0                        1
##    foundation_type.h foundation_type.i foundation_type.r foundation_type.u
## 1                  0                 0                 1                 0
## 2                  0                 0                 1                 0
## 3                  0                 0                 1                 0
## 4                  0                 0                 1                 0
## 5                  0                 0                 1                 0
## 6                  0                 0                 1                 0
## 7                  0                 0                 1                 0
## 8                  0                 0                 0                 0
## 9                  0                 0                 1                 0
## 10                 0                 1                 0                 0
##    foundation_type.w roof_type.n roof_type.q roof_type.x ground_floor_type.f
## 1                  0           1           0           0                   1
## 2                  0           1           0           0                   0
## 3                  0           1           0           0                   1
## 4                  0           1           0           0                   1
## 5                  0           1           0           0                   1
## 6                  0           1           0           0                   1
## 7                  0           1           0           0                   0
## 8                  1           0           1           0                   0
## 9                  0           0           1           0                   1
## 10                 0           1           0           0                   0
##    ground_floor_type.m ground_floor_type.v ground_floor_type.x
## 1                    0                   0                   0
## 2                    0                   0                   1
## 3                    0                   0                   0
## 4                    0                   0                   0
## 5                    0                   0                   0
```

```
## 6                    0                   0                   0
## 7                    0                   0                   1
## 8                    0                   1                   0
## 9                    0                   0                   0
## 10                   0                   1                   0
##    ground_floor_type.z other_floor_type.j other_floor_type.q other_floor_type.s
## 1                    0                   0                   1                   0
## 2                    0                   0                   1                   0
## 3                    0                   0                   0                   0
## 4                    0                   0                   0                   0
## 5                    0                   0                   0                   0
## 6                    0                   0                   1                   0
## 7                    0                   0                   1                   0
## 8                    0                   0                   0                   0
## 9                    0                   0                   1                   0
## 10                   0                   1                   0                   0
##    other_floor_type.x position.j position.o position.s position.t
## 1                   0          0          0          0          1
## 2                   0          0          0          1          0
## 3                   1          0          0          0          1
## 4                   1          0          0          1          0
## 5                   1          0          0          1          0
## 6                   0          0          0          1          0
## 7                   0          0          0          1          0
## 8                   1          0          0          1          0
## 9                   0          0          0          1          0
## 10                  0          0          0          1          0
##    plan_configuration.a plan_configuration.c plan_configuration.d
## 1                     0                    0                    1
## 2                     0                    0                    1
## 3                     0                    0                    1
## 4                     0                    0                    1
## 5                     0                    0                    1
## 6                     0                    0                    1
## 7                     0                    0                    1
## 8                     0                    0                    0
## 9                     0                    0                    1
## 10                    0                    0                    1
##    plan_configuration.f plan_configuration.m plan_configuration.n
## 1                     0                    0                    0
## 2                     0                    0                    0
## 3                     0                    0                    0
## 4                     0                    0                    0
## 5                     0                    0                    0
## 6                     0                    0                    0
## 7                     0                    0                    0
## 8                     0                    0                    0
## 9                     0                    0                    0
## 10                    0                    0                    0
##    plan_configuration.o plan_configuration.q plan_configuration.s
## 1                     0                    0                    0
## 2                     0                    0                    0
## 3                     0                    0                    0
## 4                     0                    0                    0
```

```
## 5                     0                     0                     0
## 6                     0                     0                     0
## 7                     0                     0                     0
## 8                     0                     0                     0
## 9                     0                     0                     0
## 10                    0                     0                     0
##    plan_configuration.u legal_ownership_status.a legal_ownership_status.r
## 1                     0                        0                        0
## 2                     0                        0                        0
## 3                     0                        0                        0
## 4                     0                        0                        0
## 5                     0                        0                        0
## 6                     0                        0                        0
## 7                     0                        0                        0
## 8                     1                        0                        0
## 9                     0                        0                        0
## 10                    0                        0                        0
##    legal_ownership_status.v legal_ownership_status.w
## 1                         1                        0
## 2                         1                        0
## 3                         1                        0
## 4                         1                        0
## 5                         1                        0
## 6                         1                        0
## 7                         1                        0
## 8                         1                        0
## 9                         1                        0
## 10                        1                        0
```

0.6.2 Model 1: Decision tree using rpart package

The rpart is a decision tree algorithm, it finds solutions by making sequential, hierarchical decision about the outcomes variable based on the predictor data. The maths works by splitting the dataset recursively until a predetermined termination criterion is reached. This splitting is based on the the degree of heterogeneity of the leaf nodes (split points). For example, a leaf node that contains a single class is homogeneous (also called impurity=0) and need not split. Entropy and Gini are the most common quantification method for measuring the heterogeneity. Rpart uses Gini as a default which is what we have done. Method "class"is chosen here as this relate to our predicted variable which is discrete (damage_grade).

Based on the complexity factor(cp) plot report see Figure (8) , we choose cp pf 0.00005 for final model to be run.The results show around 72% overall accuracy, however it also shows poor performance for less prevalent class particularly in predicting damage grade 1. We will try to look at other models and see whether this improves with other models.

```
## 84.96 sec elapsed
```

	F1_Class_1	F1_Class_2	F1_Class_3	Accuracy	Timetaken
Decision Tree using rpart	0.5068493	0.7765279	0.655418	0.7191589	84.96

The results from rpat decision tree algorithm shows overall accuracy of approximately 72% however it performs poorly for less prevalent class i.e. damage grade 1. Information from fitted model was also used to get the list of variables that were dominant in terms of prediction power.This is shown in figure (9.

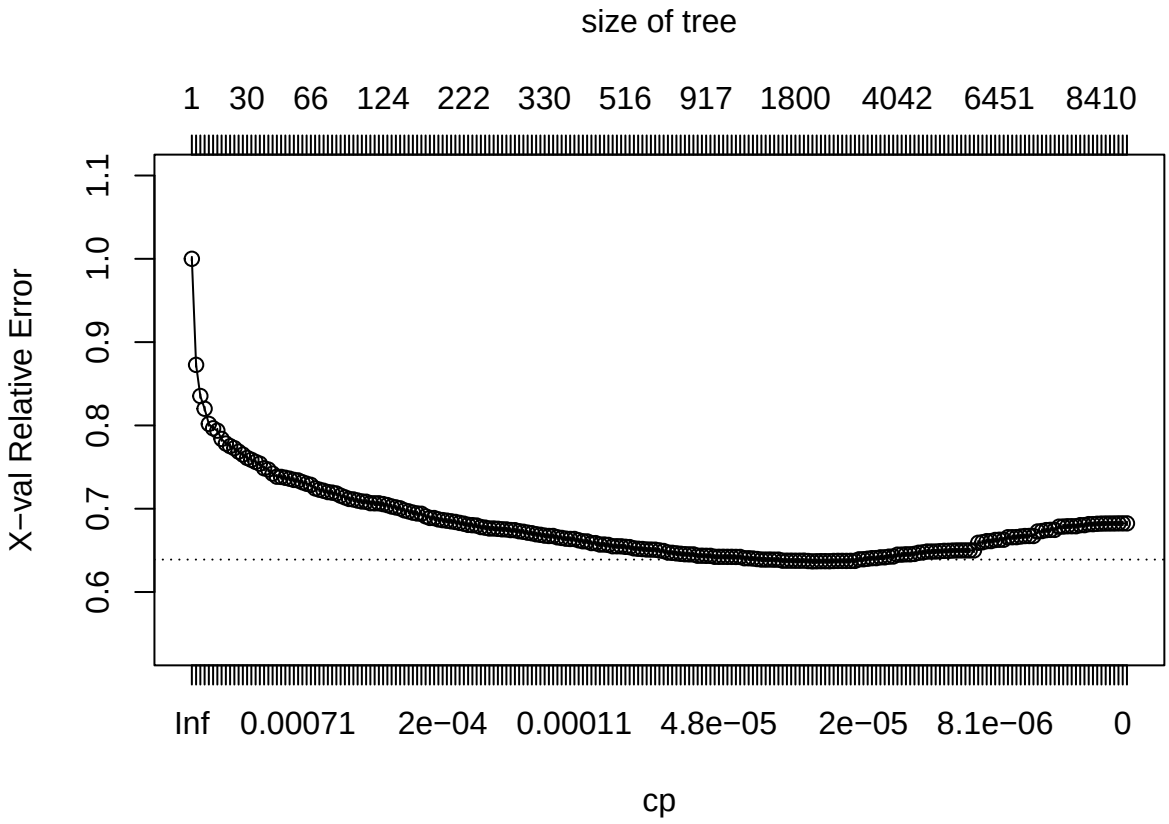


Figure 8: Cp Plot: tuning complexity factor

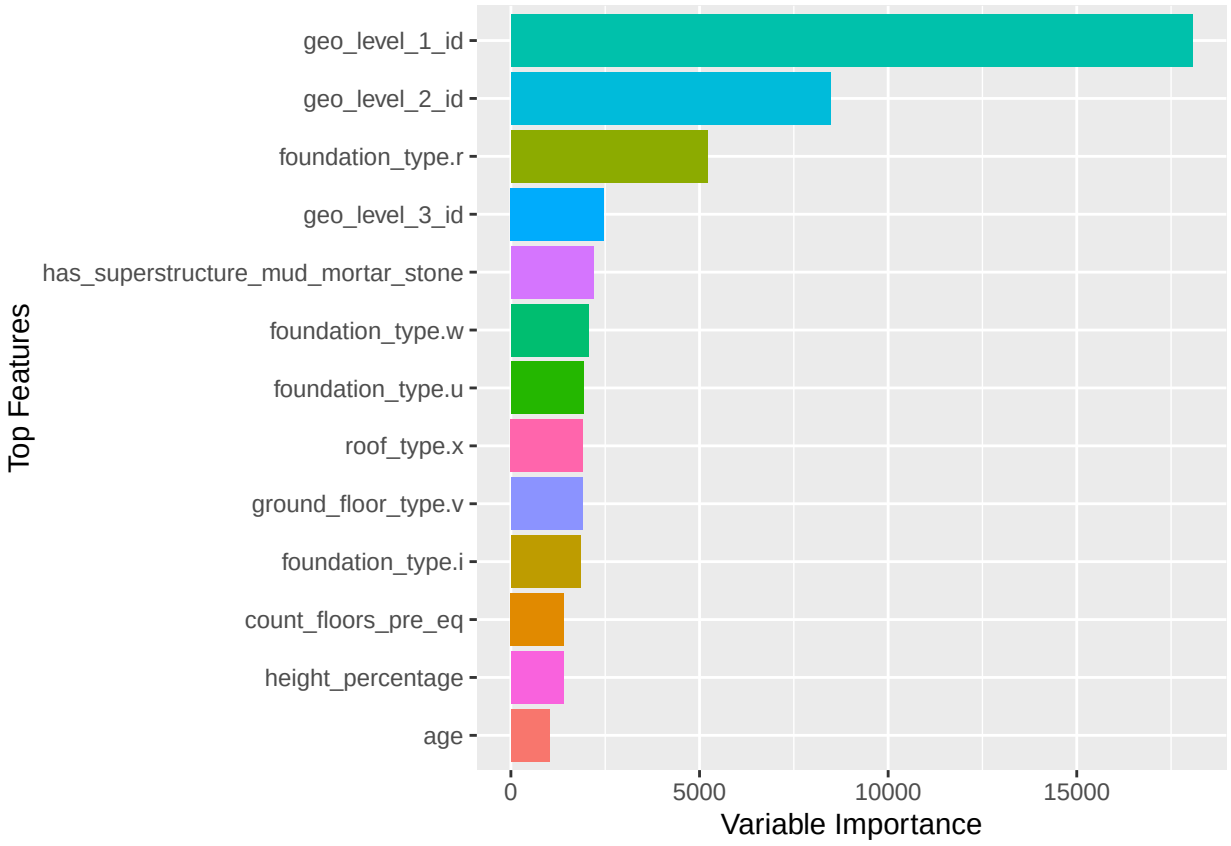


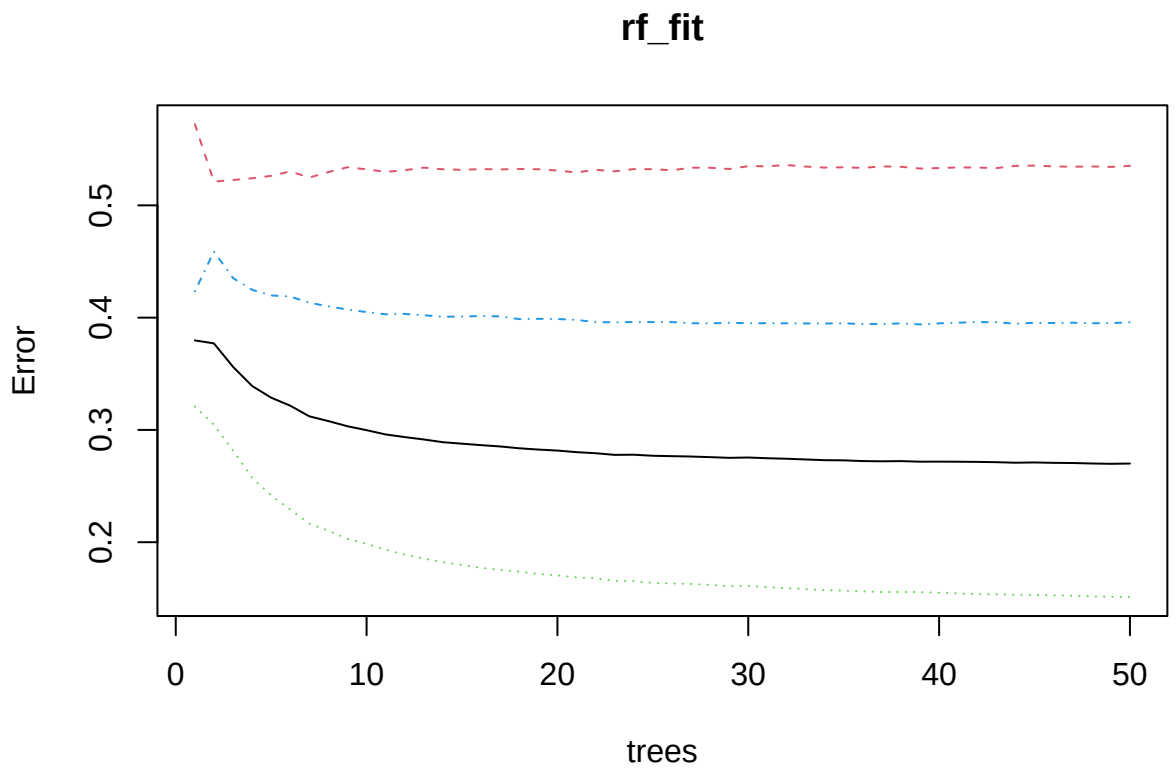
Figure 9: Importance Variables for decision tree model

0.6.3 Model 2: Random Forest

The random forest is an ensemble decision tree algorithm. It uses bagging (randomly samples with replacement) and chooses random features(n features) when building each individual tree, this way it tries create an uncorrelated forest of trees. Each tree then has a say in the final voting, the prediction is done by majority vote. Here, we choose 50 trees and only sample 50000 rows to save time. Mtry represents subset of features that is chosen for each decision tree, this is a tuning parameter but often found in the literature as square root of number of features. In our case, that is 8. However, after tuning we found mtry of >16 is optimal, similarly number of trees greater than 20 didn't make a lot of difference. The tuning took a long time and therefore, the codes have not been included here.

[Tuning mtry for random forest] (https://github.com/AnilG80/CYO_HarvardX/blob/main/mtrytuning.png?raw=true)

```
## 268.22 sec elapsed
```




```
##                                 F1_Class_1 F1_Class_2 F1_Class_3  Accuracy
## Random Forest using randomForest  0.5444629  0.7812015   0.662975 0.7264111
##                                 Timetaken
## Random Forest using randomForest    268.22
```

	F1_Class_1	F1_Class_2	F1_Class_3	Accuracy	Timetaken
Decision Tree using rpart	0.5068493	0.7765279	0.655418	0.7191589	84.96
Random Forest using randomForest	0.5444629	0.7812015	0.662975	0.7264111	268.22

The results from random forest algorithm shows overall accuracy of approximately 73% which is better than rpart algorithm. However, this also performed poorly classifying damage grade 1. variable improtance for this model was comparable to previous model, see figure (10.

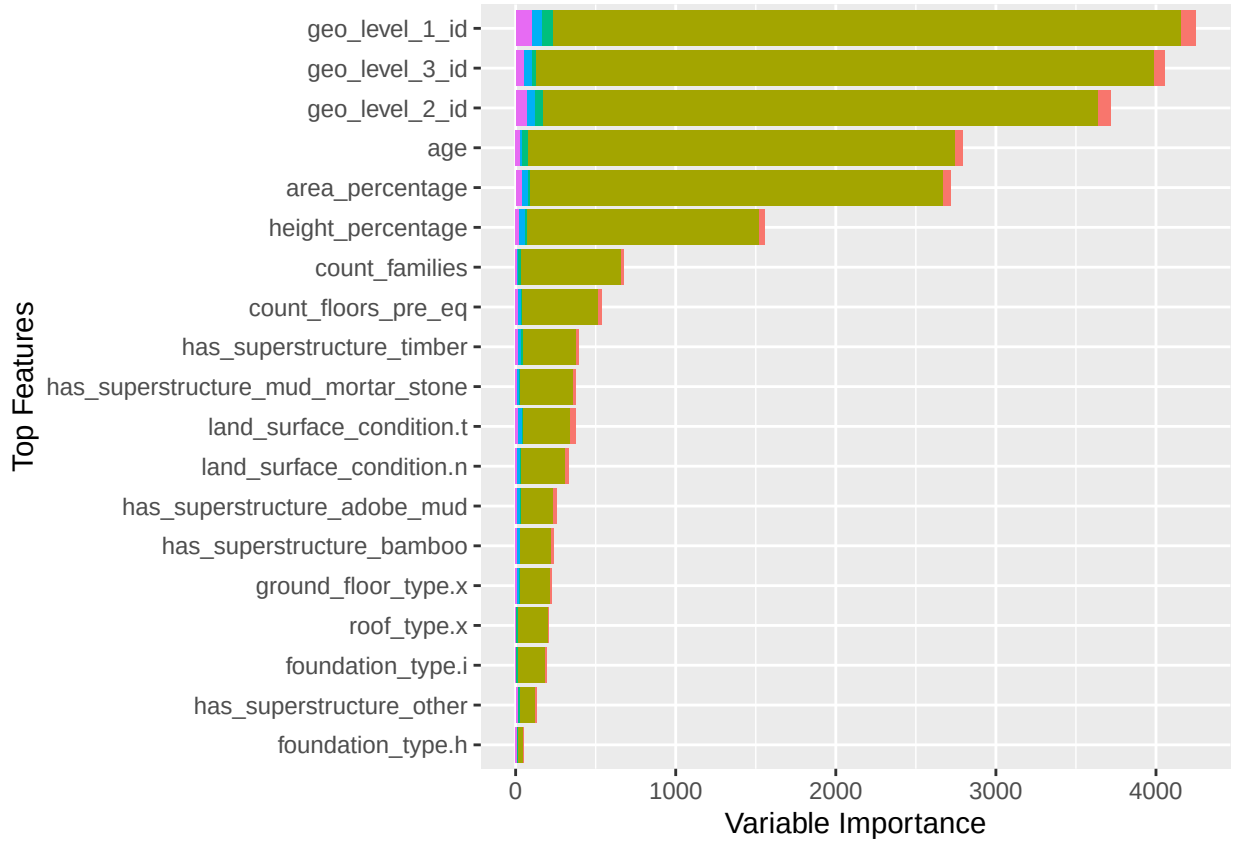


Figure 10: Importance Variables for random forest model

0.6.4 Model 3: Random Forest using only Important variables

Important variables as shown in figure (10) were used for second random forest model. This was done to see whether this would result in less time-consuming but equally performing model.

```
## 80.11 sec elapsed
```

	F1_Class_1	F1_Class_2	F1_Class_3	Accuracy	Timetaken
Decision Tree using rpart	0.5068493	0.7765279	0.655418	0.7191589	84.96
Random Forest using randomForest	0.5444629	0.7812015	0.662975	0.7264111	268.22
Random Forest with select feature	0.5471478	0.7817813	0.667512	0.7278692	80.11

As expected, the results show similar accuracy but almost four times reduction in execution time.

0.6.5 Model 4: Gradient boost using lightgbm

Gradient boosting is also based on decision tree type algorithm, the gradient comes from how the derivative of loss function against the predicted value (Gradient Descent Method) is used to minimizes the loss function,

y = ax+b+e. Boosting refers to how the algorithm tries to boost the weak learners into stronger ones. Boosting builds the first learner on the training dataset to predict the samples, then it calculates the loss (difference between observed and the predicted0, this loss or weakness is used to build an improved learner in the second stage. In each iteration, the new residual is used for the subsequent iteration. Lightgbm algorithm makes gradient boosting lot more efficient by using intelligent sampling, feature bundling and identifying splitting points using histogram-based algorithm.

The parameters used for the algorithm are detailed below:

learning_rate = 0.1 : multiplication performed on each boosting iteration. num_iterations = 1000 : how many total iterations if not early stopped objective = "multiclass" : classification model for three damage grades metric = "multi_error" : metric of choice for evaluation on the test set num_class = 3 : how many levels to classify early_stopping_rounds = 50 : number of maximum iterations without improvements. min_sum_hessian_in_leaf = 0.1 : prune by minimum hessian requirement (performed better than minimum in a leaf) num_threads = 4 : computing max_depth = -1 : maximum depth of each trained tree, -1 is unlimited boosting = "gbdt" : boosting method gradient boosting decision tree max_bin = 16320 : number of maximum unique values per feature. num_leaves = 127 : maximum leaves for each trained tree

```
## [LightGBM] [Warning] Auto-choosing col-wise multi-threading, the overhead of testing was 0.049744 se
## You can set 'force_col_wise=true' to remove the overhead.
```

```
## 56.15 sec elapsed
```

	F1_Class_1	F1_Class_2	F1_Class_3	Accuracy	Timetaken
Decision Tree using rpart	0.5068493	0.7765279	0.6554180	0.7191589	84.96
Random Forest using randomForest	0.5444629	0.7812015	0.6629750	0.7264111	268.22
Random Forest with select feature	0.5471478	0.7817813	0.6675120	0.7278692	80.11
Gradient Boost using lightgbm	0.5895743	0.7937985	0.6893911	0.7446760	56.15

The results show almost 74.5% overall accuracy with improved performance for class 1 prediction as well. This model also took the least amount of time.

0.6.6 Model 5: Extreme gradient boost using xboost

The main difference between lightgbm and xboost is how they grow decision trees. Xgboost applies level-wise tree growth based on branch level loss whereas lightgbm applies leaf-wise tree growth based on global loss. Level-wise approach grows horizontal whereas leaf-wise grows vertical trees with more depth (risk of over fitting).

```
## 120.69 sec elapsed
```

	F1_Class_1	F1_Class_2	F1_Class_3	Accuracy	Timetaken
Decision Tree using rpart	0.5068493	0.7765279	0.6554180	0.7191589	84.96
Random Forest using randomForest	0.5444629	0.7812015	0.6629750	0.7264111	268.22
Random Forest with select feature	0.5471478	0.7817813	0.6675120	0.7278692	80.11
Gradient Boost using lightgbm	0.5895743	0.7937985	0.6893911	0.7446760	56.15
Gradient Boost using xgboost	0.5702422	0.7889811	0.6788284	0.7373086	120.69

Surprisingly, xgboost didn't perform better than lightgbm. Although we could only tune eta, other parameters could only be tuned manually. This was less than ideal due to time it took for running these models.Further investigation is warranted regarding tuning these models.

0.6.7 Model 6: Lightgbm with sample weights

As we saw most of the previous model performed poorly for less prevalent class (damage grade 1), sample weighing was introduced to address this issue. Weight index was generated using a simple

```
## [LightGBM] [Warning] Auto-choosing col-wise multi-threading, the overhead of testing was 0.039844 se
## You can set ‘force_col_wise=true‘ to remove the overhead.
```

```
## 51.98 sec elapsed
```

	F1_Class_1	F1_Class_2	F1_Class_3	Accuracy	Timetaken
Decision Tree using rpart	0.5068493	0.7765279	0.6554180	0.7191589	84.96
Random Forest using randomForest	0.5444629	0.7812015	0.6629750	0.7264111	268.22
Random Forest with select feature	0.5471478	0.7817813	0.6675120	0.7278692	80.11
Gradient Boost using lightgbm	0.5895743	0.7937985	0.6893911	0.7446760	56.15
Gradient Boost using xgboost	0.5702422	0.7889811	0.6788284	0.7373086	120.69
Lightgbm with sample weights	0.6141511	0.7781097	0.7033704	0.7380377	51.98

The results show by performing sample weighing, there is an increase in accuracy in picking up less prevalent class.

0.6.8 Model 7: xgboost with sample weights

```
## [1]  train-merror:0.246728
## [2]  train-merror:0.238810
## [3]  train-merror:0.231495
## [4]  train-merror:0.226100
## [5]  train-merror:0.220972
## [6]  train-merror:0.217189
## [7]  train-merror:0.212727
## [8]  train-merror:0.208514
## [9]  train-merror:0.204237
## [10] train-merror:0.201436
## [11] train-merror:0.198426
## [12] train-merror:0.195549
## [13] train-merror:0.192828
## [14] train-merror:0.190128
## [15] train-merror:0.187300
## [16] train-merror:0.185109
## [17] train-merror:0.182710
## [18] train-merror:0.180397
## [19] train-merror:0.178585
## [20] train-merror:0.176743
## [21] train-merror:0.174834
## [22] train-merror:0.173350
## [23] train-merror:0.171617
## [24] train-merror:0.170406
## [25] train-merror:0.169174
## [26] train-merror:0.167635
## [27] train-merror:0.166590
## [28] train-merror:0.165066
## [29] train-merror:0.163476
## [30] train-merror:0.161719
```

```
## [31] train-merror:0.160355
## [32] train-merror:0.159292
## [33] train-merror:0.157596
## [34] train-merror:0.156374
## [35] train-merror:0.155269
## [36] train-merror:0.154443
## [37] train-merror:0.153749
## [38] train-merror:0.153093
## [39] train-merror:0.152007
## [40] train-merror:0.150742
## [41] train-merror:0.149571
## [42] train-merror:0.148303
## [43] train-merror:0.147339
## [44] train-merror:0.146276
## [45] train-merror:0.145639
## [46] train-merror:0.144993
## [47] train-merror:0.144035
## [48] train-merror:0.143151
## [49] train-merror:0.142090
## [50] train-merror:0.141040

## 110.52 sec elapsed

## ##### xgb.Booster
## raw: 40.3 Mb
## call:
##   xgb.train(params = params, data = dtrain, nrounds = nrounds,
##     watchlist = watchlist, verbose = verbose, print_every_n = print_every_n,
##     early_stopping_rounds = early_stopping_rounds, maximize = maximize,
##     save_period = save_period, save_name = save_name, xgb_model = xgb_model,
##     callbacks = callbacks, eta = 0.2, max_depth = 15, eval_metric = "merror",
##     objective = "multi:softprob", num_class = 3, nthread = 4)
## params (as set within xgb.train):
##   eta = "0.2", max_depth = "15", eval_metric = "merror", objective = "multi:softprob", num_class = "
## xgb.attributes:
##   niter
## callbacks:
##   cb.print.evaluation(period = print_every_n)
##   cb.evaluation.log()
## # of features: 68
## niter: 50
## nfeatures : 68
## evaluation_log:
##     iter train_merror
##        1     0.246728
##        2     0.238810
## ---
##       49     0.142090
##       50     0.141040

## Confusion Matrix and Statistics
##
##           Reference
```

```
## Prediction     1     2     3
##          1  1474   829    92
##          2   981 11698  2695
##          3    58  2299  5935
##
## Overall Statistics
##
##                Accuracy : 0.7332
##                  95% CI : (0.7277, 0.7385)
##     No Information Rate : 0.5689
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.514
##
##  Mcnemar's Test P-Value : 3.189e-11
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3
## Sensitivity           0.58655   0.7890   0.6805
## Specificity           0.96089   0.6728   0.8641
## Pos Pred Value        0.61545   0.7609   0.7158
## Neg Pred Value        0.95610   0.7073   0.8432
## Precision             0.61545   0.7609   0.7158
## Recall                0.58655   0.7890   0.6805
## F1                    0.60065   0.7747   0.6977
## Prevalence            0.09643   0.5689   0.3347
## Detection Rate        0.05656   0.4489   0.2277
## Detection Prevalence  0.09190   0.5899   0.3182
## Balanced Accuracy     0.77372   0.7309   0.7723
```

	F1_Class_1	F1_Class_2	F1_Class_3	Accuracy	Timetaken
Decision Tree using rpart	0.5068493	0.7765279	0.6554180	0.7191589	84.96
Random Forest using randomForest	0.5444629	0.7812015	0.6629750	0.7264111	268.22
Random Forest with select feature	0.5471478	0.7817813	0.6675120	0.7278692	80.11
Gradient Boost using lightgbm	0.5895743	0.7937985	0.6893911	0.7446760	56.15
Gradient Boost using xgboost	0.5702422	0.7889811	0.6788284	0.7373086	120.69
Lightgbm with sample weights	0.6141511	0.7781097	0.7033704	0.7380377	51.98
xgboost with sample weights	0.6006520	0.7747020	0.6976607	0.7331645	110.52

0.7 Conclusion and Future Work

In this work, we showed how different machine learning algorithms could be used for predicting the level of damage based on various information related to the age, location, ownership and structure of the building.

Of the four algorithms that were used in this project, lightgbm performed better both in terms of accuracy and computational efficiency. Both gradient boosting algorithms improved class 1 accuracy even without having to use sample weights, this is consistent with their algorithm that is meant to boost weak learners.

The leader board for this competition shows accuracy of 75% for top 10, although we got close to this we still have someway to go. Some of the areas for further exploration include: better algorithm for tuning parameters, using oversampling to address class imbalance and using ensemble.

0.8 Environment

R_version	SysArchitecture	System
R version 4.1.2 (2021-11-01)	x86_64	x86_64, mingw32

0.9 References

Technical details for Decision Tree (rpart) (Therneau et al. 1999 based on Friedman et al.1984) in R can be found at: https://CRAN.R-project.org/package=rpart

Technical details for ensemble decision tree (randomforest) (Breiman (2001)) in R can be found at: https://CRAN.R-project.org/package=randomForest

Technical details for xgboost (Chen et al. 2017) in R can be found at: https://CRAN.R-project.org/package=xgboost

Technical details for lightgbm (Ke, Guolin et al. (2017)) can be found at: https://CRAN.R-project.org/package=lightgbm

RStudio IDE environment details is included in the appendix. R Core Team (2020). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/.

Harirchian, E., Kumari, V., Jadhav, K., Raj Das, R., Rasulzade, S., & Lahmer, T. (2020). A machine learning framework for assessing seismic hazard safety of reinforced concrete buildings. Applied Sciences, 10(20), 7153.

Zhang, Y., Burton, H. V., Sun, H., & Shokrabadi, M. (2018). A machine learning framework for assessing post-earthquake structural safety. Structural safety, 72, 1-16.

Liang, Y. (2021). Application of Machine Learning Methods to Predict the Level of Buildings Damage in the Nepal Earthquake.