KLE Society's

KLE Technological University

A Project Report On

Richter's Predictor: Modeling Earthquake Damage

Under the guidance of

Dr P.G. Sunitha Hiremath

Submitted By

Name	USN
Nitin Verma	01FE17BCS125
Puneet Gupta	01FE17BCS144
Rahetul Asquin	01FE17BCS148
Ritbik Bharti	01FE17BCS158

SCHOOL OF COMPUTER SCIENCE & ENGINEERING,

HUBLI – 580 031 (India)

Academic year 2019-20

Content

1. Introduction

1.1 Overview of the Project

Following the 7.8 Mw Gorkha Earthquake on April 25, 2015, Nepal carried out a massive household survey using mobile technology to assess building damage in the earthquake-affected districts. This survey is one of the largest post-disaster datasets ever collected, containing valuable information on earthquake impacts, household conditions, and socio-economic-demographic statistics.

This challenge is hosted by drivendata.org and the goal is **to predict the level of damage to buildings caused by the 2015 Gorkha earthquake in Nepal.**

1.2 Problem Statement

Based on aspects of building location and construction, the goal is to predict the level of damage to buildings caused by the 2015 Gorkha earthquake in Nepal.

1.3 Problem Description

We are trying to predict the ordinal variable damage_grade, which represents a level of damage to the building that was hit by the earthquake. There are 3 grades of the damage:

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

1.4 Data Description

There are 4 files in the dataset:

1. train_values.csv
   It has training data for 260,601 houses and 39 attributes.
2. train_labels.csv
   It has damage_grade for all the training data.
3. test_values.csv
   It has test data for 86,868 houses and 39 attributes.
4. sumission_format.csv
   It has the format specified for the file to be submitted.

1.4 Features Description

The dataset mainly consists of information on the buildings' structure and their legal ownership. Each row in the dataset represents a specific building in the region that was hit by the Gorkha earthquake.

There are 39 columns in this dataset, where the `building_id` column is a unique and random identifier. The remaining 38 features are described in the section below. Categorical variables have been obfuscated random lowercase ascii characters.

- `geo_level_1_id`, `geo_level_2_id`, `geo_level_3_id` (type: int): geographic region in which building exists, from largest (level 1) to most specific sub-region (level 3). Possible values: level 1: 0-30, level 2: 0-1427, level 3: 0-12567.
- `count_floors_pre_eq` (type: int): number of floors in the building before the earthquake.
- `age` (type: int): age of the building in years.
- `area_percentage` (type: int): normalized area of the building footprint.
- `height_percentage` (type: int): normalized height of the building footprint.
- `land_surface_condition` (type: categorical): surface condition of the land where the building was built. Possible values: n, o, t.
- `foundation_type` (type: categorical): type of foundation used while building. Possible values: h, i, r, u, w.
- `roof_type` (type: categorical): type of roof used while building. Possible values: n, q, x.
- `ground_floor_type` (type: categorical): type of the ground floor. Possible values: f, m, v, x, z.
- `other_floor_type` (type: categorical): type of constructions used in higher than the ground floors (except of roof). Possible values: j, q, s, x.
- `position` (type: categorical): position of the building. Possible values: j, o, s, t.

- `plan_configuration` (type: categorical): building plan configuration. Possible values: a, c, d, f, m, n, o, q, s, u.
- `has_superstructure_adobe_mud` (type: binary): flag variable that indicates if the superstructure was made of Adobe/Mud.
- `has_superstructure_mud_mortar_stone` (type: binary): flag variable that indicates if the superstructure was made of Mud Mortar - Stone.
- `has_superstructure_stone_flag` (type: binary): flag variable that indicates if the superstructure was made of Stone.
- `has_superstructure_cement_mortar_stone` (type: binary): flag variable that indicates if the superstructure was made of Cement Mortar - Stone.
- `has_superstructure_mud_mortar_brick` (type: binary): flag variable that indicates if the superstructure was made of Mud Mortar - Brick.
- `has_superstructure_cement_mortar_brick` (type: binary): flag variable that indicates if the superstructure was made of Cement Mortar - Brick.
- `has_superstructure_timber` (type: binary): flag variable that indicates if the superstructure was made of Timber.
- `has_superstructure_bamboo` (type: binary): flag variable that indicates if the superstructure was made of Bamboo.
- `has_superstructure_rc_non_engineered` (type: binary): flag variable that indicates if the superstructure was made of non-engineered reinforced concrete.
- `has_superstructure_rc_engineered` (type: binary): flag variable that indicates if the superstructure was made of engineered reinforced concrete.
- `has_superstructure_other` (type: binary): flag variable that indicates if the superstructure was made of any other material.
- `legal_ownership_status` (type: categorical): legal ownership status of the land where building was built. Possible values: a, r, v, w.
- `count_families` (type: int): number of families that live in the building.
- `has_secondary_use` (type: binary): flag variable that indicates if the building was used for any secondary purpose.

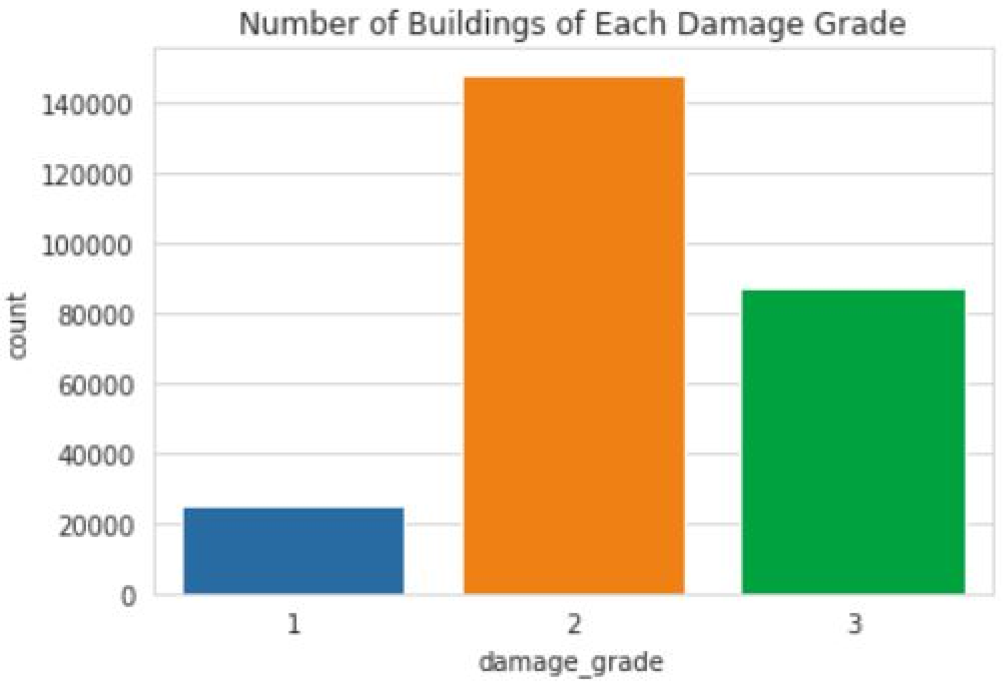
- `has_secondary_use_agriculture` (type: binary): flag variable that indicates if the building was used for agricultural purposes.
- `has_secondary_use_hotel` (type: binary): flag variable that indicates if the building was used as a hotel.
- `has_secondary_use_rental` (type: binary): flag variable that indicates if the building was used for rental purposes.
- `has_secondary_use_institution` (type: binary): flag variable that indicates if the building was used as a location of any institution.
- `has_secondary_use_school` (type: binary): flag variable that indicates if the building was used as a school.
- `has_secondary_use_industry` (type: binary): flag variable that indicates if the building was used for industrial purposes.
- `has_secondary_use_health_post` (type: binary): flag variable that indicates if the building was used as a health post.
- `has_secondary_use_gov_office` (type: binary): flag variable that indicates if the building was used fas a government office.
- `has_secondary_use_use_police` (type: binary): flag variable that indicates if the building was used as a police station.
- `has_secondary_use_other` (type: binary): flag variable that indicates if the building was secondarily used for other purposes.

2. Methodology

2.1 Exploratory Data Analysis

The datasets were relatively clean. After collecting the data and the compilation of the respective datasets into a data frame, exploratory analysis confirmed that there were **no missing values** and each variable contained the correct corresponding data types. The variables in the train values dataset contained both numeric and categorical data. To get consistent datatypes of the features, the construction of dummy variables from the categorical data were done.

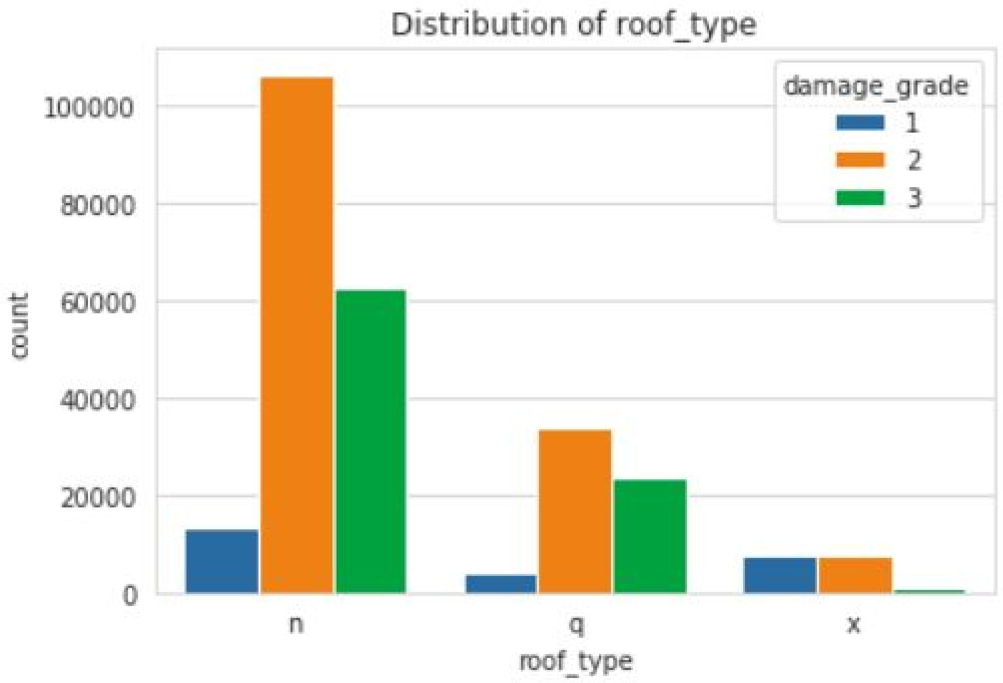
Then we looked at the distribution of the train labels.



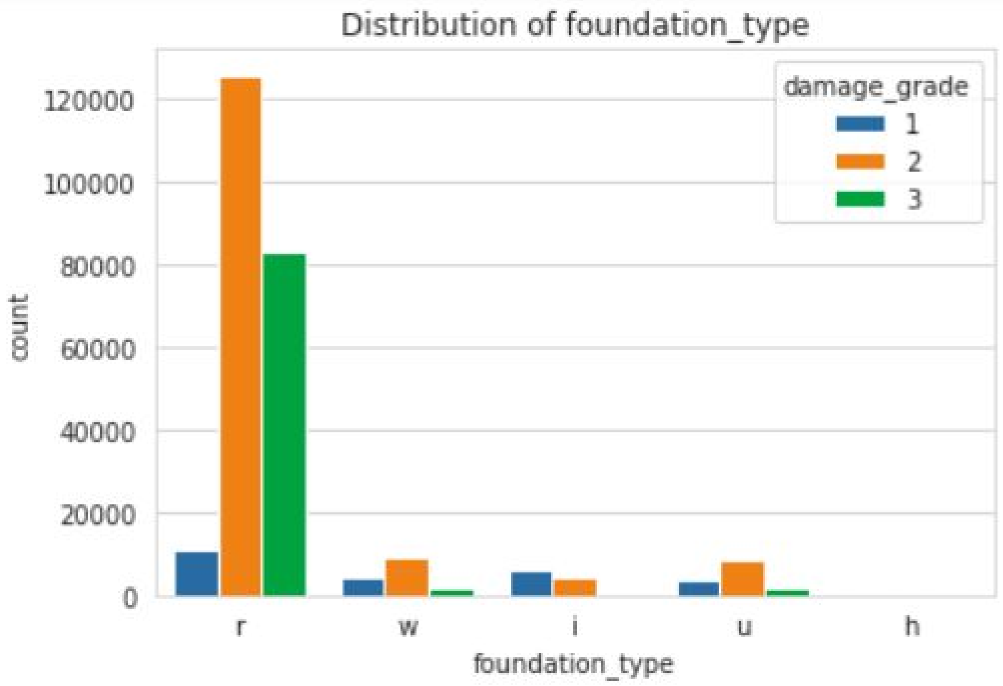
The damage grade at level 2 is noticeably of the outcomes, with more than 50%.

Exploring the numerical data first, we see that the ranges of the features differ. Some features are id, which goes up to 12,567, this is a drastic number compared to the count of floors which only goes up to 9.

Next, I explore the categorical features in the dataset. There seems to be a major disparity within each categorical feature where one value holds the majority.



The above fig shows a clear difference in roof type n and the other types of roofing.



The foundation type feature shows an even greater gap were type r composes of nearly all the foundation being used and type h has virtually none.

2.2 Pre Processing

To get consistent datatypes of the features, the construction of dummy variables from the categorical data were done.

We removed the building id feature as it was only there to identify the buildings, note playing any role in their damage.

As analysed there is less number of floors greater than 5, so we replaced higher floors with 5.

Very few families have count > 4. Hence, we replaced greater counts with 4.

We normalised the area_percentage attribute as it has a big range from 0 to 100. Mathematically that's a small range, but there is a big difference in houses built on 1 percent of area and another one built on even 50 percent of area.

We normalised the height_percenatage attribute also as almost values are near to 5 but the upper range is 32. So we wanted to bring down the scaling.

Feature selection is a process where we automatically select those features in our data that contribute most to the prediction variable.

Having too many irrelevant features in the data can decrease the accuracy of the models. Three benefits of performing feature selection before modeling our data are:

- Reduces Overfitting:
  Less redundant data means less opportunity to make decisions based on noise.
- Improves Accuracy:
  Less misleading data means modeling accuracy improves.
- Reduces Training Time:
  Less data means that algorithms train faster.

Important features by ExtraTreeClassifier :-

Each Decision Tree in the Extra Trees Forest is constructed from the original training sample. At each test node, Each tree is provided with a random sample of k features from the feature-set from which each decision tree must select the best feature to split the data based on some mathematical criteria (typically the Gini Index).

The feat_importanaces then help us to take the subset of training data consisting of only important features.

2.3 Model Building

2.3.1 Model Selection

There are many classifiers available to build the model.
We decided to initially build all the models and choose the one with best F1 score. Bagging Classifier using Random Forest gave us the best F1 score among all the classifiers.

Ensembling : Bagging Classifier
Hyper-Tune Bagging Classifier and Cross Validation using GridSearchCV :-
Grid search is the process of performing **hyper parameter tuning** in order to determine the optimal values for a given model. This is significant as the performance of the entire model is based on the hyper parameter values specified.

- The param_grid parameter requires a list of parameters and the range of values for each parameter of the specified estimator.
- n_estimators : (default=10) The number of base estimators in the ensemble.
- max_samples : (default=1.0) The number of samples to draw from X to train each base estimator.

Bagging Classifier gave us an accuracy of 0.74

Ensembling : XGBoost
We then tried XGboost ensembling technique:-

- Boosting is an ensemble technique where new models are added to correct the errors made by existing models. Models are added sequentially until no further improvements can be made.

- In XGBoost, we fit a model on the gradient of loss generated from the previous step. In XGBoost, we modified our gradient boosting algorithm so that it works with any differentiable loss function.

XGBoost gave us a better F1 Score of 0.7466

Hyper Tuning XGBoost:-

Parameters we hyper tuned :-
•Subsample – subsample ratio of training data
•Lambda - L2 regularization term on weights. Increasing this value will make the model more conservative.
•Alpha - L1 regularization term on weights. Increasing this value will make the model more conservative.

•learning_rate: usually between 0.1 and 0.01. It is wise to decrease the learning rate incrementally while increasing the number of trees.

HyperTuning XGB gave us an accuracy of 0.7478

Ensembling : Stacking

Stacking is a way to ensemble multiple classifications.

The point of stacking is to explore a space of different models for the same problem. The idea is that you can attack a learning problem with different types of models which are capable of learning some part of the problem, but not the whole space of the problem.

Stacking gave a boost in F1 score from 0.7478 to 0.7480

Ensembling : Voting Classifier

Voting classifier isn't an actual classifier but a wrapper for set of different ones that are trained and evaluated in parallel in order to exploit the different peculiarities of each algorithm.

Hard voting is the simplest case of majority voting. In this case, the class that received the highest number of votes will be chosen

Voting gave a boost in F1 score from 0.7480 to 0.7482

3. Experimentation

We also tried normalising the age attribute also but the accuracy decreased significantly.

We tried the inbuilt scikit learn standardisation method to standardise the data but again the accuracy decreased.

For feature selection we used Random Forest and ExtraTreeClassifier and found the ExtraTreeClassifier one to be more accurate.

We then took geo_level_ids as categorical attributes.

4. Result and Analysis

After predicting our model with the hypertuned parameters and applying ensembling, we created the output file as per submission format given and submitted on the competition site.

We received an F1 score of 0.7519 and currently ranks 17 out of 2603 participants.

5. References

1. http://drivendata.co/blog/richters-predictor-benchmark/

2. https://medium.com/@namanbhandari/extratreesclassifier-8e7fc0502c7