# Supervised Learning: Damage prediction

Marina Prieto Pech[1*†]

[1*]Machine Learning, UCLM, Paseo de la universidad, 3, Ciudad Real, 13002, Ciudad Real, España.

Corresponding author(s). E-mail(s): Marina.Prieto1@alu.uclm.es;
†These authors contributed equally to this work.

**Abstract**

This work studies the use of supervised learning to make predictions about the integrity level after the 2015 Gorkha earthquake in Nepal.

**Keywords:** Supervised Learning, Predictions, Earthquake

## 1 Introduction

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.

The data was collected through surveys by Kathmandu Living Labs and the Central Bureau of Statistics, which works under the National Planning Commission Secretariat of Nepal.

We're trying to predict the ordinal variable **damageGrade** , which represents a level of damage to the building that was hit by the earthquake.

More Info about the competition that originated this project clicking here.

## 2 Dataset Description

On this dataset we can observe the following variables:

- **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.

## 3 Performance Metric Description

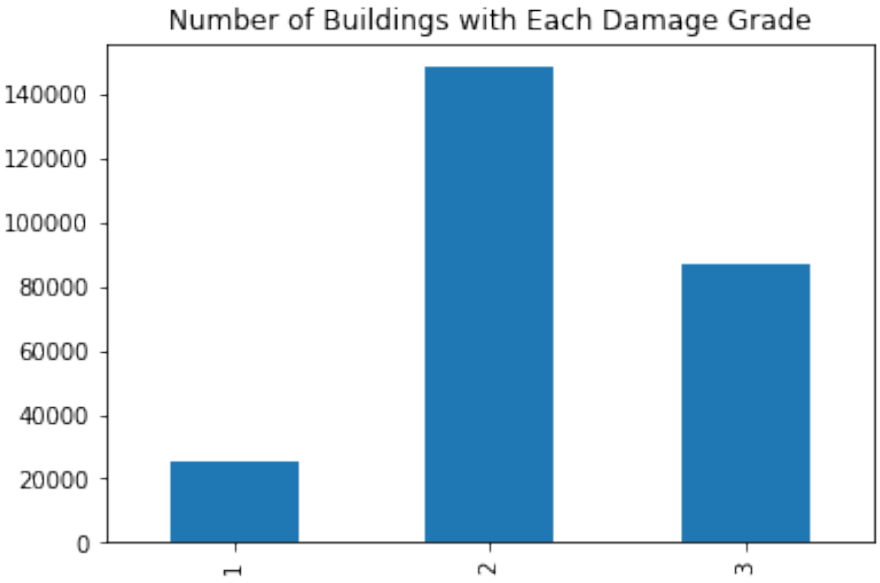
To measure the performance of our algorithms, we'll use the **F1 score** which balances the precision and recall of a classifier. Traditionally, the F1 score is used to evaluate performance on a binary classifier, but since we have three possible labels we will use a variant called the **micro averaged F1 score**.

$$F_{micro} = \frac{2 \cdot P_{micro} \cdot R_{micro}}{P_{micro} + R_{micro}}$$

$$P_{micro} = \frac{\sum_{k=1}^{3} TP_k}{\sum_{k=1}^{3} (TP_k + FP_k)}, \quad R_{micro} = \frac{\sum_{k=1}^{3} TP_k}{\sum_{k=1}^{3} (TP_k + FN_k)}$$

## 4 Starting Steps

First of all, we need to import and load all the needed datasets. We can see the exploration result in this graphic:

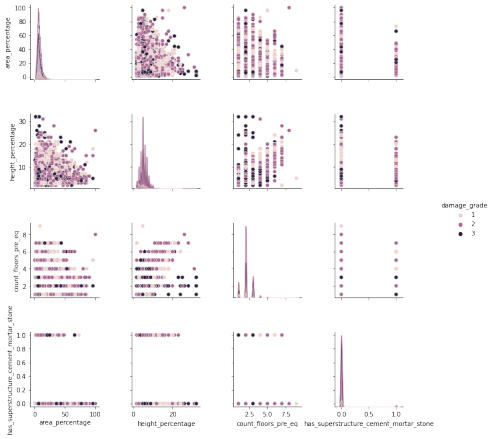



## 5 Feature Selection and Preprocessing

After the initial analysis of the dataset, we may select the dataset features we need, which will be:

- **foundation_type**
- **area_percentage**
- **height_percentage**
- **count_floors_pre_eq**
- **land_surface_condition**
- **has_superstructure_cement_mortar_stone**

For better visualization, we can see the pairplot of the data selected:



We transform non-numerical values with help of OneHotEncoding and proceed with the process.

After that, we pre-process the set by dividing it into two types: X and Y, applied to the train and test datasets separately.

# 6 Data Modelling

Now we proceed with the main part, which is creating the training model for the predictions. For this, we are going to divide into various sections.

## 6.1 Modelling using NaiveBayes

For this we need to preevaluate the F1Score value. We can also see the metrics obtained which would be:

| | Precision | Recall | F1-Score | Support |
|---|---|---|---|---|
| 1 | 0.25 | 0.64 | 0.36 | 6238 |
| 2 | 1.0108298 | 0.4418198 | 0.15 | 37210 |
| 3 | 1.845912 | 0.4779288 | 0.52 | 21703 |
| accuracy | | | 0.37 | 65151 |
| macro avg. | 0.42 | 0.50 | 0.34 | 65151 |
| weighted avg. | 0.52 | 0.37 | 0.29 | 65151 |

Now we just need to create the confusion matrix after that for our model and study to be done.

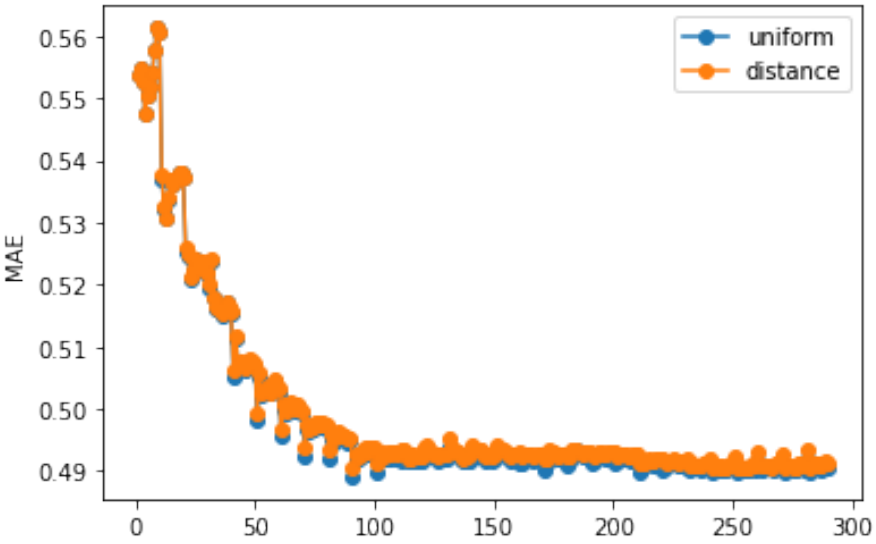
## 6.2 Modelling using kNN

For this modelling process we first need to reset the indexes from all the previous actions before starting.

### 6.2.1 Training execution

We perform the euclidean metric for kNN and complete their steps: training, testing, training and execution, and finally evaluating the model. This whole proccess may take a while depending on the device in which it is executed.

We end up having our min. values uniform and min. value of distances, being 0.4891... and 0.4904... respectively.



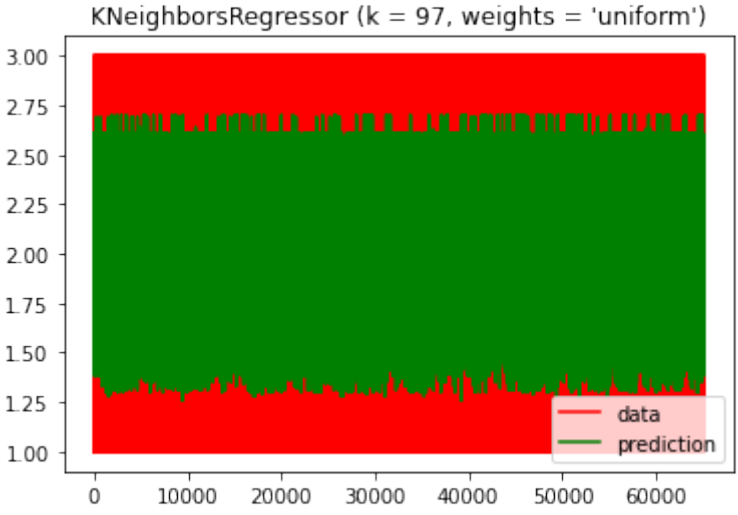

### 6.2.2 Prediction

For the predictions, we execute the regression model (we concluded the number of neighbours we're taking will be 97 according to the graphic results), fit that model into the training values and with that form our prediction.

On the .ipynb file we can observe all the error metrics as well

### 6.2.3 Visualization

To finish off the model and see our results, we can plot the data-to-prediction relationship. This results in the next graphic:

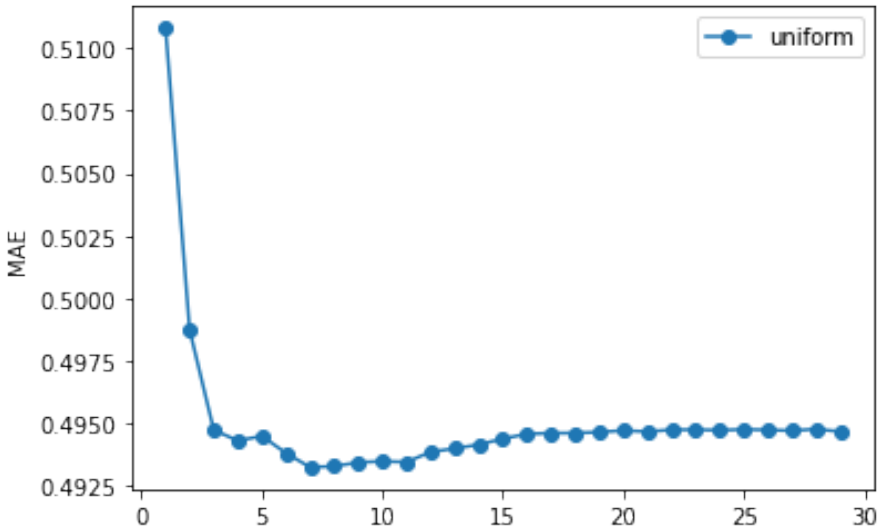


## 6.3 Decision Trees

Other way of prediction is with the help of decision trees.

### 6.3.1 Training execution

We divide the train and test values randomly, train the model and evaluate it to finish the process. We end up with a Min. value uniform of 0.4932... and the following graphic visualization:




### 6.3.2 Model construction

We construct the Decision Tree Regressor model taking a maximum depth value of 7, according to the previous graphic results.

After that, we can simply execute the model and construct the tree.

### 6.3.3 Prediction

To end up with the decision tree, we just obtain the error metrics and the decision tree visual representation. Both available at the file.

## 6.4 Submission Formatting

To end up all the process and submit our results to the platform, we just get the predictions formatted into the dataFrame format that is given, and export them as a .csv file.

## 6.5 Submission Page

All the information for this project was obtained through DrivenData., and access to my profile can be done by clicking Here