Machine Learning Techniques Supervised Learning

Francisco Jesús Díaz Pellejero[†], Tamara Redondo Soto[†] and Javier Villar Asensio[†]

Contributing authors: FcoJesus.Diaz@alu.uclm.es; Tamara.Redondo@alu.uclm.es; Javier.Villar@alu.uclm.es;
[†]These authors contributed equally to this work.

Abstract

Use of machine learning techniques to predict the level of damage of an hypothetical earthquake in Nepal using the data of 2015 Gorkha earthquake. We see that the more complex the technique used, the better prediction we obtain.

1 Introduction

We have been given a large dataset with data of the 2015 Gorkha earthquake in Nepal. With that dataset we have been asked to find the best prediction method using supervised learning techniques. In order to know that our prediction is useful we have used the f1 score and the percentage of accuracy.

The dataset has 38 features for each building that was evaluated after the earthquake and each one its corresponding level of damage being the lowest 1 and the highest 3. Along our work we have been making little improvements every step so eventually we would have the best prediction, this consisted on choosing the mos important features and using more complex techniques.

2 Preparing the data

As is mentioned, data is given as a table of 260601 rows (that refers to every building analyzed) with 38 columns each (every feature considered). The first thing we notice is that not every feature is represented by a numerical value (which are the values we need) so we will have to turn them into numbers.

Table 1 Features

Name of feature	Type of data
geo level 1 id	int64
geo level 2 id	int64
geo level 3 id	int64
count floors pre eq	int64
age	int64
area percentage	int64
height percentage	int64
land surface condition	object
foundation type	object
roof type	object
ground floor type	object
other floor type	object
position	object
plan configuration	object
has superstructure adobe mud	int64
has superstructure mud mortar stone	int64
has superstructure stone flag	int64
has superstructure cement mortar stone	int64
has superstructure mud mortar brick	int64
has superstructure cement mortar brick	int64
has superstructure timber	int64
has superstructure bamboo	int64
has superstructure rc non engineered	int64
has superstructure rc engineered	int64
has superstructure other	int64
legal ownership status	object
count families	int64
has secondary use	int64
has secondary use agriculture	int64
has secondary use hotel	int64
has secondary use rental	int64
has secondary use institution	int64
has secondary use school	int64
has secondary use industry	int64
has secondary use health post	int64
has secondary use gov office	int64
has secondary use use police	int64
has secondary use other	int64

If the type says object, it means that is and String object, a character.

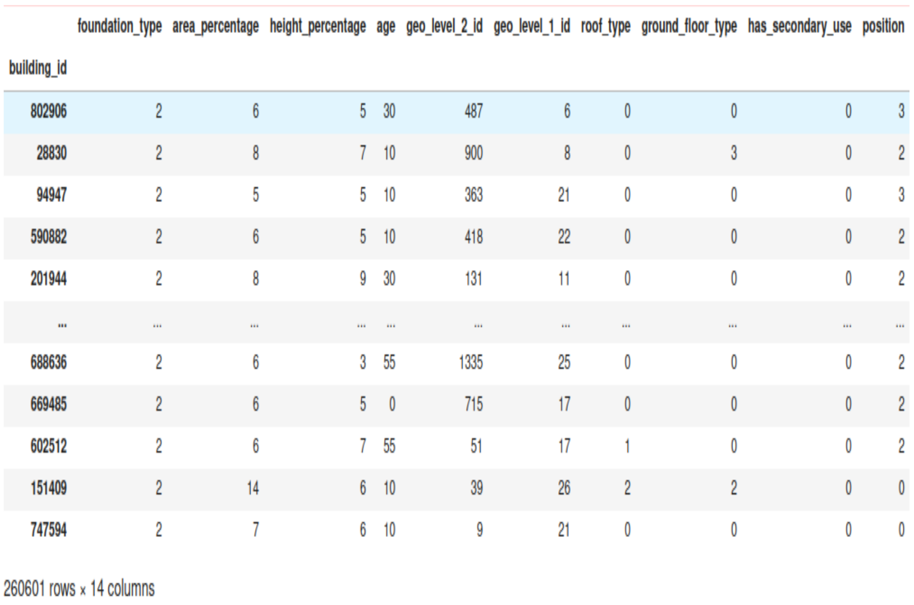
We have chosen the features that we considered the most important, this has not been a simple step because we have tested the algorithms several times so we could choose the features with more importance. Despite the selective election, later, you may see that there are some features much more important then the rest, but we have kept less important features because if you add the importance of the weaker ones you still can gain up to a 20% of importance. The features we did not use are hardly relevant in importance.

```
foundation_type
area_percentage
height_percentage
age
geo_level_2_id
geo_level_1_id
roof_type
ground_floor_type
has_secondary_use
position
plan_configuration
has_superstructure_mud_mortar_stone
has_superstructure_timber
has_superstructure_bamboo
```

Later we will see the importance of each feature in the different techniques. The criteria used was taking the features that could be more affecting in an earthquake. To sum up the features, the represent where the building is located, the method of construction and some characteristics that may affect itself.

So we join the tables of the features and the table with the damage grade in and we have our database with the whole dataset. As it is said, once we have our database we convert the String values to numbers so our data table is finally prepared.
[1]

[1] As the dimensions are too large the table is partially represented



building_id	foundation_type	area_percentage	height_percentage	age	geo_level_2_id	geo_level_1_id	roof_type	ground_floor_type	has_secondary_use	position
802906	2	6	5	30	487	6	0	0	0	3
28830	2	8	7	10	900	8	0	3	0	2
94947	2	5	5	10	363	21	0	0	0	3
590882	2	6	5	10	418	22	0	0	0	2
201944	2	8	9	30	131	11	0	0	0	2
...	...	...	...	...	...	...	...	...	...	...
688636	2	6	3	55	1335	25	0	0	0	2
669485	2	6	5	0	715	17	0	0	0	2
602512	2	6	7	55	51	17	1	0	0	2
151409	2	14	6	10	39	26	2	2	0	0
747594	2	7	6	10	9	21	0	0	0	0

260601 rows × 14 columns

3 Basic techniques

The following techniques use the given data in mathematical algorithms to sort every individual. First, we divide our dataset into two parts that do not need to be necessarily the same length, one part will be used to train the model and the other part will be used to use our fitted model and test it to see the results and how useful the model is.
The metric we are using is the *micro average F1 - score* being the best result **1**.

3.1 Naive Bayes

The Naive Bayes method consists of calculating the probabilities of a given building of belonging to a certain class, the probabilities are calculated with a given sample to train the model.

$$P(A1, A2, ..., An\mathsf{C}) = \mathsf{P}(\mathsf{A1}Cj)P(A2\ \mathsf{Cj})...\ \mathsf{P}(\mathsf{An}Cj) \tag{1}$$

The algorithm calculates the probability of belonging to a class **C** given the attributes **A**, thus must be calculated to every class. In this study the classes are the levels of damage and the attributes are the features we have selected. The metric result is **0.3357738177464659**. This result is very low but as we continue upgrading our techniques we will see how this value increases. See fig 1

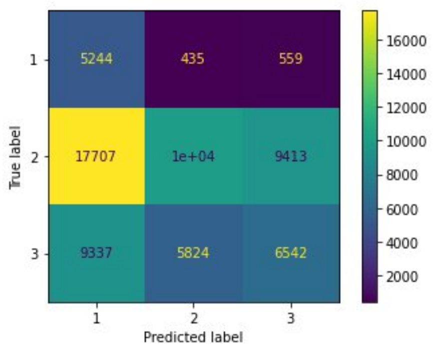
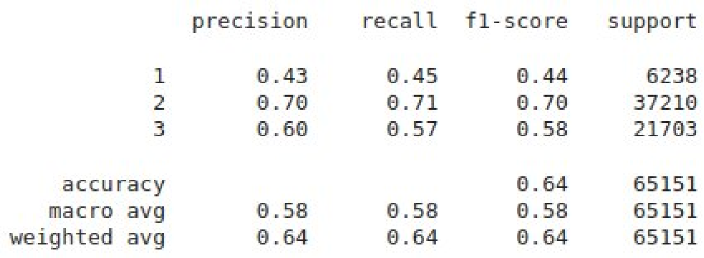


Fig. 1 We can see it is not very accurate the Naive Bayes' method



```
              precision    recall  f1-score   support

           1       0.43      0.45      0.44      6238
           2       0.70      0.71      0.70     37210
           3       0.60      0.57      0.58     21703

    accuracy                           0.64     65151
   macro avg       0.58      0.58      0.58     65151
weighted avg       0.64      0.64      0.64     65151
```

Fig. 2 Results with KNN

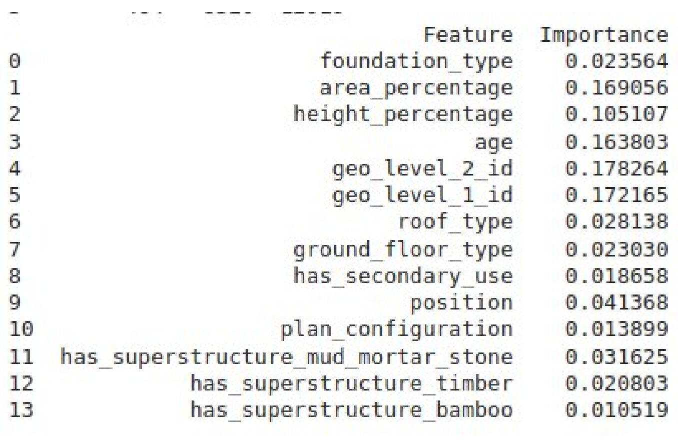
3.2 KNN

K-Nearest-Neighbours is a method in which when we evaluate an instance we check its K nearest neighbours (previously classified by the model) and set the queried instance the class that is most common in them. This method despite the fact that may it seem simple produces such good results. The metric obtained was **0.64**. See fig 2

3.3 Decision Trees

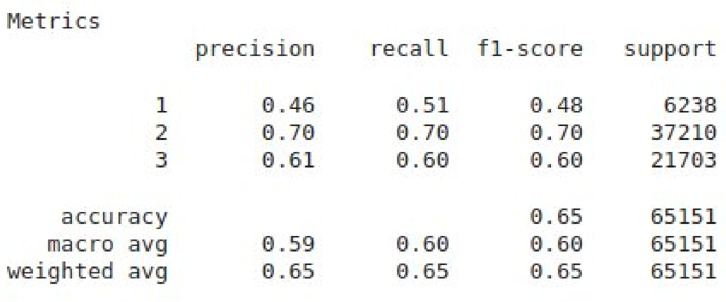
The method of the decision tree is based on the truthness of questions we do to the instance we are evaluating about its features. Those questions create a tree in which the leaves are the different classes (our three levels of damage) and every level of the tree represents a question.
With this algorithm we can see the importance of each feature because we can see which nodes of the tree did the instanced queried passed through. See fig 3

We see that the most important features are its geographical location, its dimension and its age.
Decision tree is the best method of the three basic methods used. Its metric value is **0.639867384997928**. See fig 4



```
                                  Feature  Importance
0                         foundation_type    0.023564
1                         area_percentage    0.169056
2                       height_percentage    0.105107
3                                     age    0.163803
4                          geo_level_2_id    0.178264
5                          geo_level_1_id    0.172165
6                               roof_type    0.028138
7                       ground_floor_type    0.023030
8                       has_secondary_use    0.018658
9                                position    0.041368
10                     plan_configuration    0.013899
11  has_superstructure_mud_mortar_stone    0.031625
12             has_superstructure_timber    0.020803
13             has_superstructure_bamboo    0.010519
```

Fig. 3 Importance of the features in the decision tree



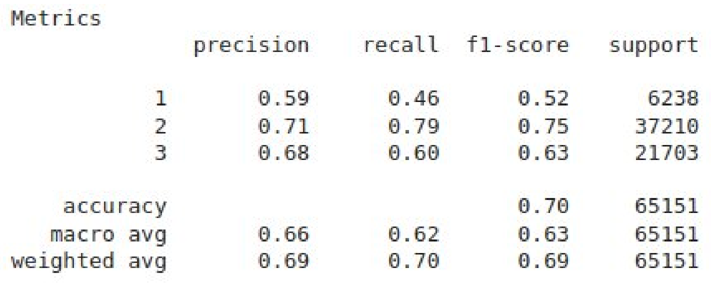
```
Metrics
              precision    recall  f1-score   support

           1       0.46      0.51      0.48      6238
           2       0.70      0.70      0.70     37210
           3       0.61      0.60      0.60     21703

    accuracy                           0.65     65151
   macro avg       0.59      0.60      0.60     65151
weighted avg       0.65      0.65      0.65     65151
```

Fig. 4 Results of the prediction with the decision tree

4 Ensembles

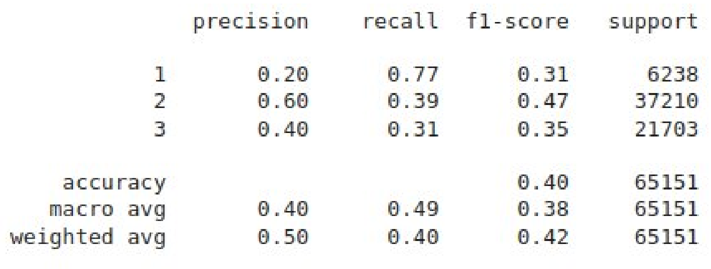
The main problem of the basic algorithms we have used previously is that it may be weak with new data and their error is still a bit high, to solve this, we run multiple instances of the method so they mix up to form a stronger model. One example of these methods can be applied to the decision trees, we run multiple decision trees and we mix them up to form the best decision trees. We only have to decide the number of instances, for this study we have used 100. This way, the results improve. See fig 5

We can improve even more this model optimizing the hyperparameters. The hyperparameters are the parameters used by the model regardless of the parameters of the database. This optimization is similar to how ensembles works, multiple instances of the model are tested with different values in the parameters and we used the one with the best results.
The forest decision trees with this optimizations is called *Randomized forest classifier*. The new metric value has increased to **0.710794922564504**



```
Metrics
               precision    recall  f1-score   support

           1       0.59      0.46      0.52      6238
           2       0.71      0.79      0.75     37210
           3       0.68      0.60      0.63     21703

    accuracy                           0.70     65151
   macro avg       0.66      0.62      0.63     65151
weighted avg       0.69      0.70      0.69     65151
```

Fig. 5 Results with the forest tree



```
               precision    recall  f1-score   support

           1       0.20      0.77      0.31      6238
           2       0.60      0.39      0.47     37210
           3       0.40      0.31      0.35     21703

    accuracy                           0.40     65151
   macro avg       0.40      0.49      0.38     65151
weighted avg       0.50      0.40      0.42     65151
```

Fig. 6 Little improvement with GS in Naive Bayes

5 GridSearch

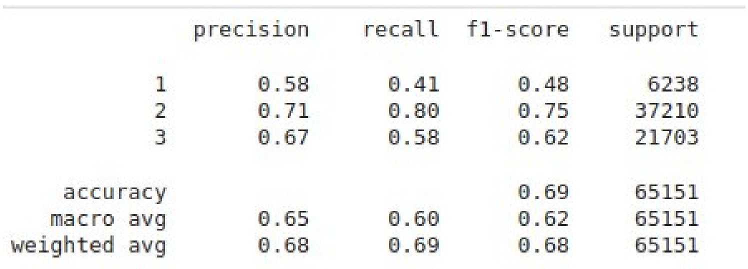
The aim of this search is to upgrade the hyperparameters of the model to use the best option, the algorithm accepts a range of values for the hyperparameters given by the programmer and ends up using the best combination to a given model. This is not a model whose target is prediction but it is to wrap a prediction algorithm which hyperparameters can be optimized.
The improvement we get with grid search might seem to be little but remember that our results are better the more little improvements we make.

5.1 GS with Naive Bayes

Naive Bayes is a special case to use this method as it only has 1 hyperparameter, *alpha*, but we can still run the search with multiple values of alpha. See fig 6
The metric has increased **9** points

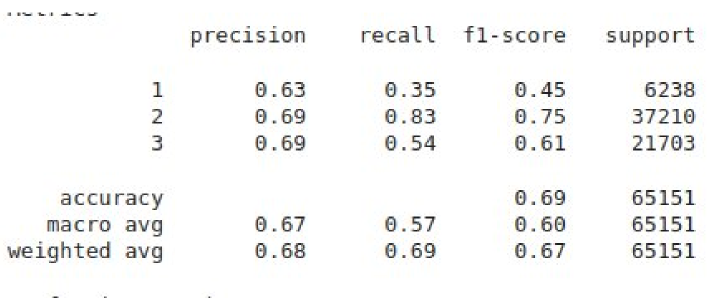
5.2 GS with KNN

From now on, with the rest of the methods we will not have the problem we had with Naive Bayes because the rest of the methods have multiple hyperparameters to be optimized. In KNN the parameters we optimized are the number of nearest neighbours to compare and the weight they have. In every



```
              precision    recall  f1-score   support

           1       0.58      0.41      0.48      6238
           2       0.71      0.80      0.75     37210
           3       0.67      0.58      0.62     21703

    accuracy                           0.69     65151
   macro avg       0.65      0.60      0.62     65151
weighted avg       0.68      0.69      0.68     65151
```

Fig. 7 Improvements with GS in KNN



```
              precision    recall  f1-score   support

           1       0.63      0.35      0.45      6238
           2       0.69      0.83      0.75     37210
           3       0.69      0.54      0.61     21703

    accuracy                           0.69     65151
   macro avg       0.67      0.57      0.60     65151
weighted avg       0.68      0.69      0.67     65151
```

Fig. 8 Improvements with GS in decission tree

grid searchs we get improvements. See fig 7.
The metric has increased **4** points

5.3 GS with Decission Tree

In the decission tree the hyperparameters optimized are; the criterion used to rate the tree as the *gini* (which is a way to mark the tree) or its *entropy*, the max depth of the tree in a range from 1 to 10, the minimun number of samples to split a node and the minimun samples per leaf. See fig 8.
The metric has increased **2** points.

5.4 GS with Forest Decission Trees

The hyperparameters we have optimized are; the maximun number of features considered in every split, the maximun depth of the tree and the minimun of samples per leaf and per split using ranges. The ranges had to be very low numbers due to the time needed to execute the algorithm, this will affect its results.
The result we got is worse than without the GS due to the threshold we had to put and the metric value is **0.5780724777823825**

6 Competition Results

Here we have a summary of the *f1 score* we had in each method in table 2

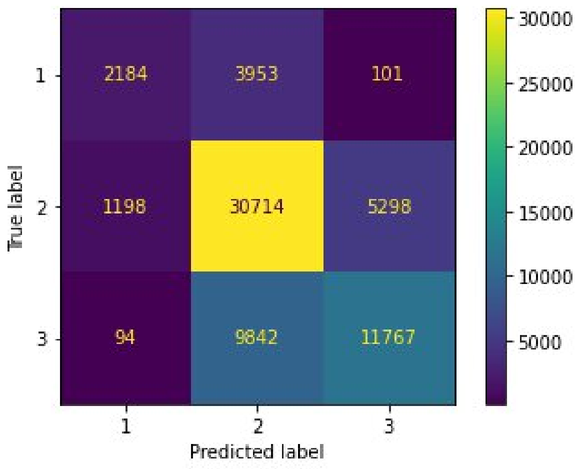


Fig. 9 Heat map of the decission tree with GS

Table 2 f1 scores

Method	f1 score
Naive Bayes	0.33
KNN	0.64
Decission Tree	0.63
Forest Decission Tree	0.69
Randomized Forest Classifier	0.71
GridSearch NaiveBayes	0.42
GridSearch KNN	0.68
GridSearch Decission Tree	0.67
GridSearch Forest Decission Tree	0.57

Summary of f1 scores.

The highest one is **Randomized Forest Classifier** with **0.71** *f1 score*

7 Conclussions/Summary

Having all the data collected and the results we can evaluate. As we have said previously, many times in machine learning are the little improvements we get in every step that builds up a good solution.
It all started with the selection of the features, the final features used is not the first selection, as we have the possibility of consulting which were the mos important features in, for example, the decission tree algorithm, we have tested several times with different features, not only with different test but with a little understanding in how earthquakes works some features had more likelihood of being chosen, such as the location of the building (the nearest to the epicenter, the more energy it suffers) or the age (the worse conditions

of the foundation, the more possibility of destruction). Then we run different basic methods and saw their results with the features we selected. And finally we used ensembles and hyperparameter optimization to improve the basic algorithms.
Unfortunately, we could not have the best results that our algorithms are capable of, this is our compute power's fault. Some of these algorithms can be much better adding more range of testing and more accurate instructions but increasing too much the time of execution. Some examples of these we had is the range of the minimun samples in a node in the decisiont tree to split, if we put a big range the execution would last a lot, and in KNN we could not put a lot of neighbours due to the same reason.
Having much more power of computation that allowed us to take advantage of the algorithms and their capabilities would show better results.