Richter's Predictor: Modeling Earthquake Damage

Carlos Pulido Hernández[1], Ismael Pérez Nieves[1†] and Sergio Jiménez Fernández[1†]

[1*]Machine Learning Techniques, Escuela Superior de Informática, Ciudad Real, Castilla-La Mancha, Spain.

Contributing authors: carlos.pulido@alu.uclm.es; ismael.perez3@alu.uclm.es; sergio.jimenez19@alu.uclm.es;
[†]These authors contributed equally to this work.

Abstract

For the scope of the second part of the Machine Learning Techniques subject, about Supervised Learning algorithms, the task consists on building a model and participate in a competition on Driven Data. By this way, given some values for certain building's structure and legal status/ownership attributes, predicts the potential damage of the 2015 Gorkha earthquake over it and classifies it in three different levels depending on its consequences over the building. In this report we present several details related with the development of the model -the applied techniques- as well as the corresponding parametrization and, of course, the accuracy score obtained according to both, the F1 Score from the Scikit Learn Python library and the F1 Score variant for the Driven Data competition.

Keywords: Machine Learning, Supervised Learning, Artificial Intelligence

1 Introduction

Back in 2015, in Nepal, took place an earthquake with a severe intensity between 7.8 and 8.1 on the Richter scale with its epicentre $81km$ northwest of Kathmandú, more specifically in the Lamjung district. It had many catastrophic consequences in several aspects [1]:

- Over 300 aftershocks with a peak intensity of 7.9 [2].
- Over 9000 deaths and 22300 injuries [1].
- Thousands of houses were destroyed [3].
- 35% of the Nepal's GDP as economic loss [1].

Some time after the horrible disaster, in the Driven Data platform, they propose a challenge: build a model that predicts the value of a categorical label (*damage_grade*) that represents the level of damage that the earthquake caused over a given building. This *damage_grade* variable can have three values: (i) Level 1, low damage; (ii) Level 2, medium amount of damage; (iii) Level 3, almost complete destruction.

Each building given as input to the model, has several attributes related with several relevant features about its construction and its legal status. For a more detailed feature description, they put at our disposal this section [4]. And this is what the task for the Supervised Learning part for the Machine Learning Techniques subject is about.

In this report, we first expose the employed materials and methods (Section 2). Then, in Section 3 we explain with more details the different iterations (Subsections 3.1, **??**, 3.3 and 3.4) over the project until reaching the solution for the problem. And finally, some conclusions are given in Section 4 and some additional materials are included in the appendices A, B and the users with which the submissions were made are also detailed in C.

2 Materials and Methods

2.1 Data Selection

The data used for this task was provided by the Driven Data platform included in this competition [4]. It includes:

- **Submission format**, in order to upload models in an standardized way.
- **Test Values**, with the corresponding entries for testing the model.
- **Train Labels**, includes the labels for the training data.
- **Train Values**, with the corresponding entries to work with for crafting the model.

2.2 Exploratory Data Analysis

The first step we did before even starting with the feature selection was to try to obtain some properties of the target label (*damage_grade*).

As we can see in Figure 1, there is a clear predominance of label 2, which leads to unbalanced data, i.e., the probability of a building being classified (randomly) as class 2 is not the same as the probability of being classified as 1 or 3, for example. In other words, there is not a uniform distribution of the target label.

This is an issue to take into consideration since for further algorithms, some parameters might vary in order to take this correctly into account.

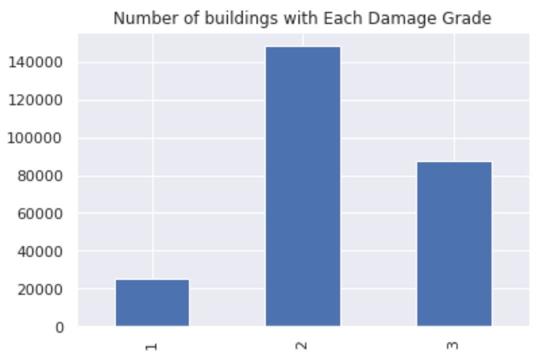

Fig. 1 Number of appearances of each label in the train labels data-set.

2.3 Methods

For model development, we had to iterate several times over the baseline. First, we made an initial selection of features based on our knowledge criteria in order to optimize the results of the model, except for decision trees, which used the whole variables for further variable impact analysis.

Then, with the results of decision trees, we considered it appropriate to iterate once again but this time, with those features that had a considerable importance rate and apply it again on some of the algorithms.

Finally, once our baseline is properly established, we will move on with a third iteration, including the use of complex (ensemble) models and even a fourth iteration trying to apply hyperparametrization aiming to maximize the final score of the model.

3 Results

Now, we expose the results of applying the previously exposed methodology. For more deeper details and results for any of the subsections exposed in this part, we present our Google Colab notebook A with the implementation and for further information, our GitHub Repository B in the appendix of this report.

3.1 1^{st} Iteration

3.1.1 Feature Selection

Once we had analysed on a first view the target label for further considerations when building the model, we did a feature selection aiming to increase the accuracy of the future models. As we have seen in class several times, when building a machine learning model, a critical step is to make a feature selection over the given data, since not all the features might be needed depending on each model.

For the first iteration of the baseline, we decided to select some features according to our knowledge criteria. Then, we gave this feature selection as input to the first technique used, Naive Bayes, and check the accuracy (according to the F1 score, from Scikit Learn package). After this initial step, we removed/added some variables also based on our knowledge criteria, taking as checkpoint the initial feature selection. A total of 14 tests were done, considering the different models provided (ComplementNB, BernoulliNB, GaussianNB and MultinomialNB) and the nature of the features. Finally, we took those variables that maximize the accuracy with Naive Bayes algorithm.

According to the previously explained process, the final feature selection for the first iteration was:

$count\_floors\_pre\_eq$, age, $ground\_floor\_type$, $has\_secondary\_use$, $land\_surface\_condition$, $roof\_type$, $foundation\_type$, $height\_percentage$, $has\_superstructure\_adobe\_mud$, $has\_superstructure\_mud\_mortar\_stone$, $has\_superstructure\_stone\_flag$, $has\_superstructure\_cement\_mortar\_stone$, $has\_superstructure\_mud\_mortar\_brick$, $has\_superstructure\_cement\_mortar\_brick$, $has\_superstructure\_timber$, $has\_superstructure\_bamboo$, $has\_superstructure\_rc\_non\_engineered$, $has\_superstructure\_rc\_engineered$, $has\_superstructure\_other$

We can see that almost all the selected features are binary, so in this case, we used a BernoulliNB model, which fits data that are distributed according to the multivariate Bernoulli distribution. That is, the features are assumed to be a binary-valued variable.

Once the first feature selection was performed, the next step was to start studying and trying the different algorithms and techniques to establish the best possible baseline.

3.1.2 Techniques

Now that we have obtained the feature selection that maximizes the accuracy of Naive Bayes, we apply kNN.

For this purpose, we used Cross Validation to determine the goodness of a particular k-value, and so, choose the best model. After analyzing the different k-values that were tested, we concluded that the best value was $k = 128$ neighbors.

Once we executed the algorithm with $k = 128$ neighbors, we obtained a f1_score of 0.5801 for the competition. Taking into account that we are applying plain kNN, it is a very good result for the first iteration.

Then, we tried with Decision Trees. In this case, we have selected all the features because this algorithm does not only provides a classification method, it also allows us to check the importance rate of each feature.

First, we need to perform an optimization process, also known as Tuning, to optimize some parameters, which in our case, is only the max_depth. The result obtained from this process, was that

Finally, we executed the algorithm and obtained a f1_score of 0.6482 for the competition, which is way better than the one obtained from kNN. As we said, Decision Trees allow us to check the importance rate of the selected features. This way, we obtained the top six features with the most importance:

$$geo\_level\_1\_id,\ geo\_level\_2\_id,\ geo\_level\_3\_id,\ foundation\_type,$$
$$has\_superstructure\_mud\_mortar\_stone,\ age$$

Considering these features, we decided to iterate again and check whether this new feature selection makes us getting a better baseline model.

3.1.3 Summary

As we can see in Table 1, the models for the first iteration are not quite high (regarding the competition leading scoreboard), but taking into account that we are still in the first iteration, it is not bad neither.

Table 1 Iteration 1 Models f1 Score

Iteration 1	
Algorithm	F1_Score
Naive Bayes	0.5052
kNN	0.5801
Decision Trees	0.6482

But wise, further improvements are expected in the following iteration of the baseline with the new feature selection based on the Decision Tree variable ranking.

3.2 2^{nd} Iteration

3.2.1 Feature Selection

For this second iteration, the feature selection was based on the importance rate of the Decision Tree, since it is considered also a power full tool in feature selection apart from its classification capabilities.

As we explained before, in paragraph 3.1.2, we have reduced considerably the number of features, from 19 variables according to the knowledge criteria and the Naive Bayes algorithm indications to only six out of the 36 original features.

3.2.2 Techniques

In this new iteration, we decided not to execute again Naive Bayes due to our previous decision of using it as an indicator for the feature selection in the first iteration. Apart from that, we considered that iterating over kNN and Decision Trees was fair enough.

For kNN, we applied Cross Validation again, and obtained that the best value was $k = 16$ neighbors. We can highlight from this that the number of neighbors gets reduced significantly regarding the previous iteration. Then, we executed the algorithm and obtained significant improvements, getting an f1_score of 0.7040 in the competition. We should note how significantly the accuracy has grown thanks to the more precise feature selection.

In this iteration, we decided to do the Tuning process again but this time using StratifiedKField as the Cross Validation process. However, the results were very similar, so we kept using the previous values, i.e., it does not suppose enough significance to iterate again with these results.

Finally, for Decision Trees, the process is the same as the first iteration, but the results are different when changing the feature selection. The Tuning process leads to a result of $max\_depth = 23$. Once the parameters are optimized, we execute the model and obtain a f1_score of 0.6486 which improves slightly the one obtained from the first iteration.

3.2.3 Summary

Concluding this iteration, we can state that the baseline was successfully established and with a considerably better accuracy than the models from the first iteration, as we can highlight form Table 2.

Table 2 Iteration 2 Models f1 Score

Iteration 2	
Algorithm	F1_Score
kNN	0.7040
Decision Trees	0.6486

Next steps will focus on trying with several ensembling models and getting a more complex model that offers us a better performance.

Another important consideration for further iterations is that we will not change the feature selection. We consider that up from now, we found an optimal selection of variables, so the following complex models and hyperparametrization steps will work based on the feature selection presented in this second iteration (paragraph 3.1.2).

3.3 3^{rd} Iteration

3.3.1 Techniques

For our third iteration, we will use ensemble boosting models based on votation. There are several algorithms to construct weak estimators, b ut in our case we will focus, for now, in AdaBoost and XgBoost.

Our first attempt was with AdaBoost. Since AdaBoost uses simpler models sequentially to get a final result, we used it to further develop our baseline models. Our kNN model brought us the best results, so we thought about basing our AdaBoost model on it. However, the current Scikit's implementation of AdaBoost isn't capable of using kNN as base estimators since the KNeighborsClassifier class doesn't support weights as attribute, so we put this idea aside in order to focus on other ensemble models.

We then tried using XgBoost with 500 estimators. The idea behind this wasn't to get a component model, but to improve upon our previous results and have a baseline model for the hyperparametrization step. After executing the model, we obtain a f1_score of 0.699

After the implementation inconvenience with the AdaBoost algorithm, the professor proposed us to research about another ensembling methods - not based on trees- that could offer us better accuracy. We finally tried the StackingClassifier model.

The StackingClassifier is another kind of ensembling technique with the idea of combining several weak learners (of different kinds, i.e., we could perfectly combine a Naive Bayes model with a Decision Tree as base learners) to output predictions base on the individual predictions of the base learner by means of the criteria of a meta learner [5].

And finally, we have the VotingClassifier, another ensembling method proposed by the teacher. The VotingClassifier also takes a combination of baseline models, but this time the result comes from all the models voting.

There are two types of votes: hard and soft. Hard only takes into account the final decision of each model, while soft uses the predicted probability of the output class. We looked into both of them and hard voting gave us the better results.

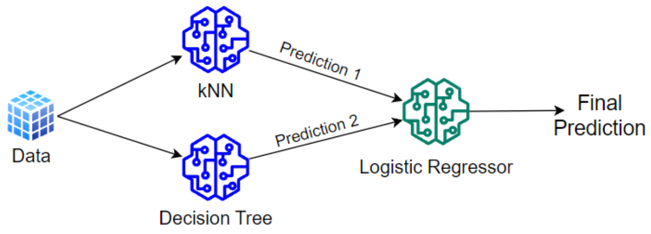

Fig. 2 Stacking Classification process.

For our application, we decided to use as base learners our top two best baseline models, according to its F1_score (Table 2), i.e. kNN and Decision Tree. Then, as the meta learner, we used the LogisticRegression algorithm, since we are in a classification problem. [6]. For a more visual explanation of the applied process, see Figure 2.

3.3.2 Summary

As a summary of this third iteration, the results (measured with the F1_score) are not considerably better than the baseline, as we can check in the Table 3.

Next -and final- improvements should focus on the hyperparametrization of the ensemble models exposed in this iteration.

3.4 4^{rd} Iteration

For the final iteration, as we previoulsy mentioned, we will try to optimize as much as possible the accuracy of our models by means of hyperparametrization

Table 3 Iteration 2 Models f1 Score

Iteration 2	
Algorithm	F1_Score
Stacking Classifier	0.7166
XgBoost	0.6990
Voting Classifier	0.7144

techniques, aiming to reach the final model to be presented as solution to the building classification problem.

Is important to remark that, in order to apply hyperparametrization, we will do it over the XgBoost tree-based ensemble method.

First of all, we tried with the RandomizedSearch Cross Validation with the parameter distribution exposed in Table 4. We obtained a final f1_score of 0.6987, which is almost as good as our best baseline, although we were warned about possible worse results when applying hyperparametrization.

Table 4 RandomizedSeachCV parameter grid

Hyperparameter	Value Range
learning_rate	[0.1, 0.01]
colsample_bytree	[0.6, 0.8, 1.0]
subsample	[0.6, 0.8, 1.0]
max_depth	[2, 3, 4]
n_estimators	[100, 200, 300, 400]
reg_lambda	[1, 1.5, 2]
gamma	[0, 0.1, 0.3]

Finally, we had to try with the remaining studied hyperparametrization algorith, GridSearch Cross Validation. In this occasion, the used parameter distribution is exposed in Table 5. After submitting this into the platform, we obtained a final score of 0.7213.

Table 5 GridSearchCV parameter grid

Hyperparameter	Value Range
n_estimators	[32, 64, 128]
max_features	['auto', 'sqrt']
max_depth	[8, 4, 2]
min_samples_split	[2, 4, 6]
min_samples_leaf	[8, 12, 16]
bootstrap	[True, False]

3.4.1 Summary

Once all the hyperparametrization techniques were applied, we can finally state that our final solution for the problem is based on using an XgBoost tree-based ensemble technique hyperparametrized with a Grid Search Cross Validation.

4 Conclusions

As conclusion to this report, we have seen how the accuracy progress have not been actually linear as we expected before starting the project. Instead, we can clearly see in Figure 3 how, in general terms, the accuracy has been growing, but not in a linear way.

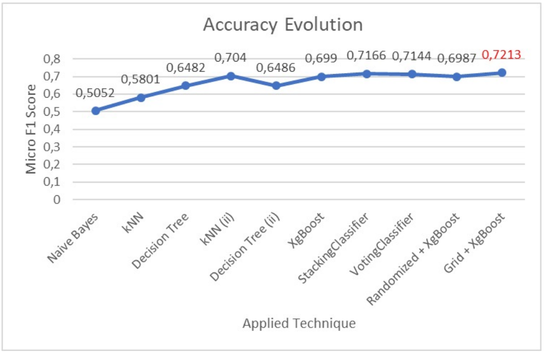

Fig. 3 Micro F1 Score Evolution of the submitted models.

Firstly, there were an initial linearity, corresponding with the first iteration models, reaching a local maximum with the kNN model of the second iteration. After that, the accuracy decreases with the other model of the second iteration. Then, with the third iteration -complex models where applied- accuracy grows again slowly until getting, with the fourth iteration -with the hyperparametrization- a global maximum for the accuracy.

So, according to the previous discussion, the final model to be proposed as solution to the problem of determining the impact in terms of damage for a given building of an earthquake with similar statistics and influence of the happened in Nepal is based on using XgBoost "hyperparametrized" by means of GridSearch Cross Validation with a $72,13\%$ of accuracy -according to the micro f1 score employed in the Driven Data platform- on its prediction.

Appendix A Google Colab Notebook

All this study was developed mainly in a Google Colaboratory Notebook. Here you can find the actual implementation and results of the different algorithms and techniques used and explained among this report.

https://colab.research.google.com/drive/1S5SfckVWHi8buZkTpBrxZAEIR2nMdnua?hl=es#scrollTo=QO52l0HSLkWn

Appendix B GitHub Repository

In the *richter* folder of the GitHub Repository, all the pertinent source code, submissions and considerations involved in the development of this project.
https://github.com/SergioJF10/MLT-ESI-UCLM_CIS

Appendix C Driven Data Users

All the three participants of the project have been uploading submissions in the platform. Here we detail the users' nicknames:

- topocart (With the best submission).
- SergioJF10
- CarlosPH

References

[1] Wikipedia: April 2015 Nepal earthquake — Wikipedia, The Free Encyclopedia. http://en.wikipedia.org/w/index.php?title=April%202015%20Nepal%20earthquake&oldid=1125169804. [Online; accessed 04-December-2022] (2022)

[2] Wikipedia: April 2015 Nepal earthquake — Wikipedia, The Free Encyclopedia. http://en.wikipedia.org/w/index.php?title=April%202015%20Nepal%20earthquake&oldid=1125169804. [Online; accessed 04-December-2022] (2022)

[3] Shrestha, S.: Langtang is gone. Nepali Times. Accessed 04-12-2022

[4] DrivenData: Richter's Predictor: Modeling Earthquake Damage. https://www.drivendata.org/competitions/57/nepal-earthquake/ Accessed 15-11-2022

[5] Rocca, J.: Ensemble Methods: Bagging, Boosting and Stacking. https://towardsdatascience.com/ensemble-methods-bagging-boosting-and-stacking-c9214a10a205 Accessed 10-12-2022

[6] Brownlee, J.: Stacking Ensemble Machine Learning With Python. https://machinelearningmastery.com/stacking-ensemble-machine-learning-with-python/ Accessed 10-12-2022