Projet Machine Learning: Final Document

Summary:

I. Description of the Project

We have chosen the topic of earthquakes because for us, it makes sense to work on a subject that touches a lot of people all around the world, and maybe helping them to have a better life in the future.

A. Context and Objectives

Living in a world with more and more natural disasters due to the climate imbalance leaves us with more incertitudes towards the future in general. In fact, even if we have precise technologies to detect an imminent earthquake, we still struggle to predict them with a length of time that will let populations to be prepared. Obviously, we can not refrain from earthquakes, but we can adapt to them. Moreover, if we can predict the precise location of an earthquake, we can also change the architecture to avoid huge damages such as the biggest earthquake in Japan on March 11th, 2011 with a magnitude of 9,1 on the Richter scale making 19,759 deaths (2,553 people missing). Still with this example, if we know the exact place of a future earthquake and its magnitude, we can adapt the architecture in packed areas such as cities with a huge population, to avoid major damages and so human disasters.

So, our main objective in this project is to try to predict earthquakes with their magnitude and location, probably using other attributes to have better predictions.

B. Formalization of the Problem

Our main problem is to predict earthquakes of different types, but we have to be precise in our predictions, in order to be legit in our work so that maybe these predictions will be useful.

The objective of our project is quite easy to understand, however, it is more complex to solve this problem than we may think.

We will use different methods to solve this problem. We can see our problem in two different ways: a regression or a classification problem. To explore the maximum of possibilities, we will try to use algorithms in both. If we see this problem as a regression problem, we will try to predict the magnitude with as much precision as possible. And if we treat it as a classification problem, we will certainly predict if the magnitude is lower or higher of a certain value. In this case, it will be pertinent to classify our prediction as if the earthquake will be considered as a micro, minor, slight, light, moderate earthquake, or as a moderate, strong, major, great, extreme earthquake (using this classification, we will take 6,0 on Richter scale as the value of reference, for example).

So, in this project, we will try as many models and algorithms as possible, to solve our problem and fulfil our objective.

Magnitude	Description
1.0–1.9	Micro
2.0–2.9	Minor
3.0–3.9	Slight
4.0–4.9	Light
5.0–5.9	Moderate
6.0–6.9	Strong
7.0–7.9	Major
8.0–8.9	Great
9.0–9.9	Extreme

Magnitude on the Richter scale and the different categories (Wikipedia)

II. Methodology

In order to fulfil our objective, we need to accomplish specific tasks. First of all, we need to find a dataset with enough data, otherwise, it will be impossible to make predictions or they will not be precise enough. In this dataset, we need at least two types of information: the location and the magnitude. It will be a plus to have the location in coordinates so that we can use it easier to show information on a map for example. Other data may be present in the dataset but we will certainly get rid of them later.

In the next step, we need to preprocess the data, which means checking if there are any missing values and if so, deleting them or replacing them by the median or something else. It also includes checking the types of the data and converting them if needed (for example, if we have categorical values, we need to convert them into numerical values).

If possible, we have to plot the data on maps, heat maps, diagrams etc. We will use the Python library matplotlib.

Then, it is time to check the correlation between the attributes in our dataset, by calculating and plotting confusion matrices.

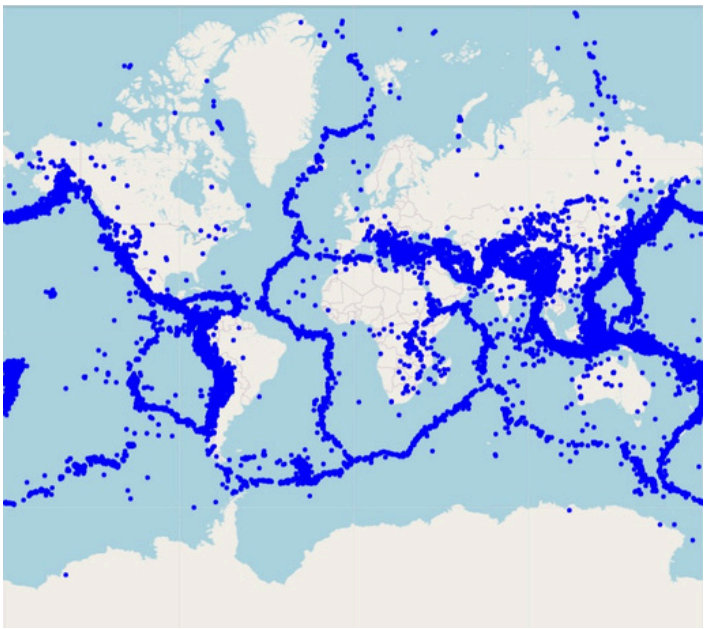
After all this preprocessing, we have to split our dataset into train and test groups so that we can implement different models and algorithms.

First, we will implement basic models such as linear regression or KNN, with maybe different heuristics. For each model, we will evaluate its performance so that we can choose the best ones at the end of the project. The next step is to use more elaborate algorithms, but we have to tune the hyperparameter, then do the feature engineering and handle class imbalance. Among the advanced algorithms we can use LightGBM regression or Deep Neural Networks (DNN) for example. And finally, we will try to implement a deep learning algorithm to solve our problem.

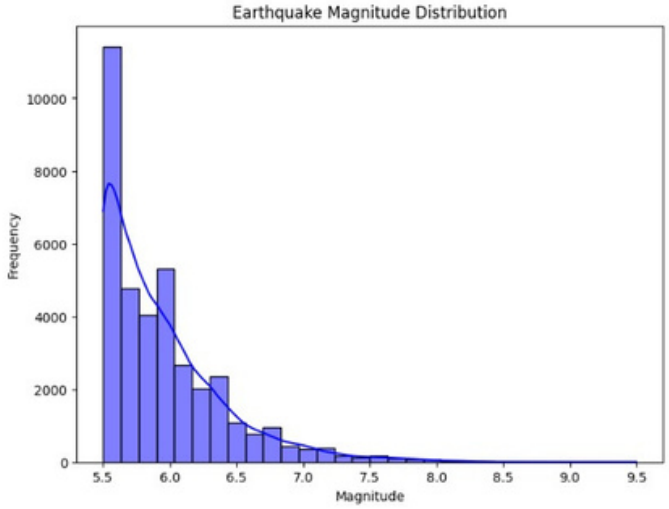
The final step will be to compare the differences between all the models and algorithms we have used and choose the best ones, comparing the accuracies of each but also the complexity of code that we have written, and the execution times.

III. Data and Results

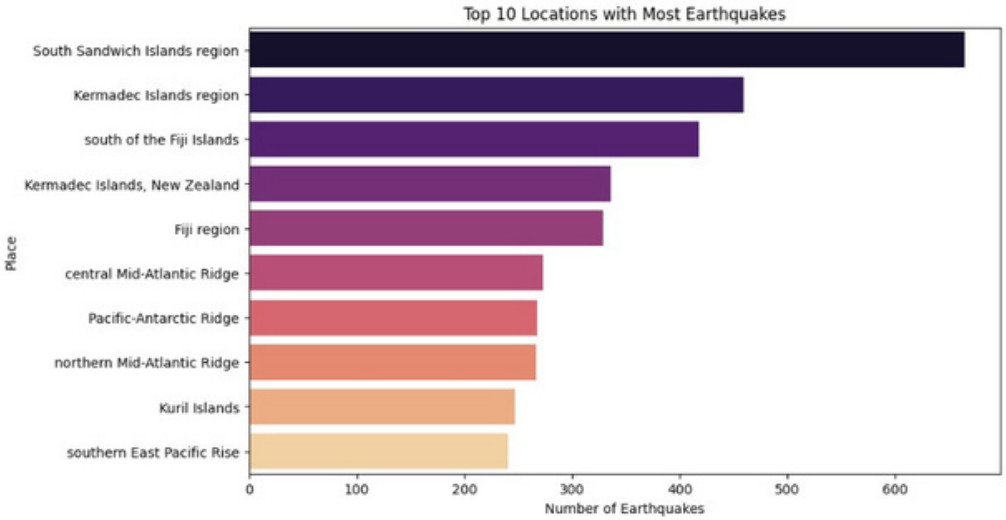
After multiple researches on Kaggle, we have found different datasets with earthquakes, focusing on Asia, with different time periods, starting from low magnitude and more. With more reflections, we have decided that we will work on a dataset including earthquakes all around the world, and not a specific area of the globe, with magnitudes of moderate class or stronger (going around 5 to 9,9 which is the maximum on the Richter scale). For the time period, we thought that the dataset with 50 years of data, from 1965 to 2016, was not the best to solve our problem, it was a too short time period. So after more research on Kaggle, we found a dataset including data from 1900 to 2023 and from all around the world, with magnitudes starting from 5,5. We finally decided to use this dataset for our project. All of those data comes from the National Earthquake Information Center (NEIC).



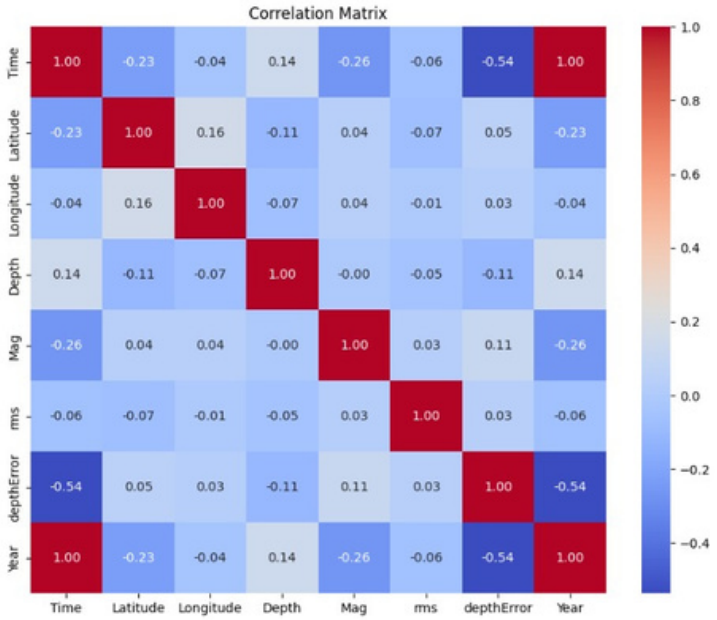
Map plot of our dataset



Earthquake magnitude distribution with our data



Plot of the top 10 locations with the biggest number of earthquakes from 1900 to 2023



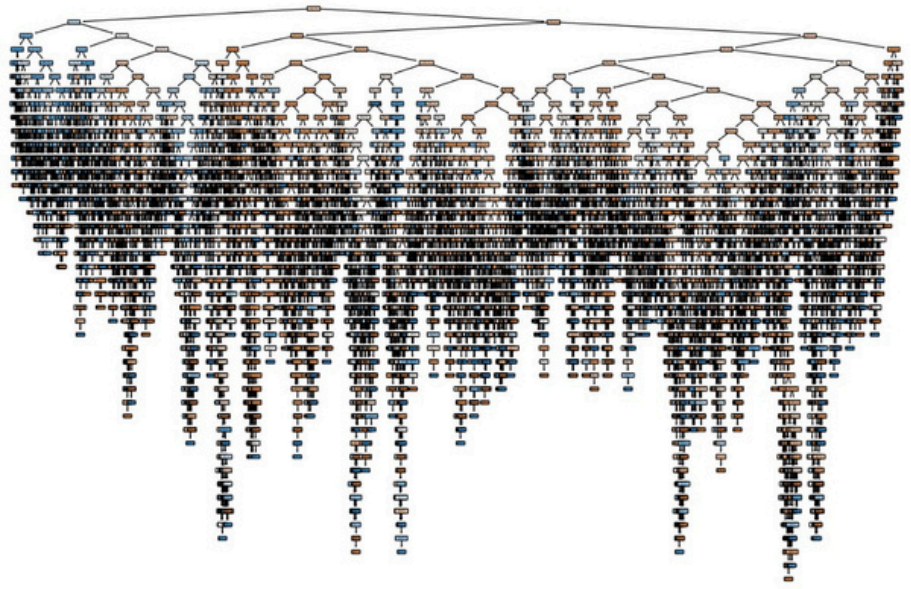
Correlation matrix of the numerical variables in our dataset

After preprocessing our data, plotting in different ways and calculating correlation matrices, we apply different models and algorithms, so let's compare them:

Model / Algorithm / Concept	Description	Accuracy	Comments
Linear regression	For the first try, we implemented the algorithm as if it was a regression problem.	(RMSE: 0.44 R2: 0.07)	Very poor results that indicate that the model is not capturing the relationship between the features and target variable effectively.
Linear regression classification	We retried the LR but as a classification problem where: class 0 represents earthquakes with mag < 0 and class 1 represents earthquakes with mag >= 6.	0,64	Works well, easy to code. We have chosen a magnitude of 6,0 as a reference because over 6,0, an earthquake is considered as "strong" and not "moderate" anymore.
Logistic regression classification	Like the previous algorithm, we tried another classification algorithm.	0,66	Better than the previous one, but still not very accurate.
Random forest classifier	Again a classification algorithm. We are keeping with this method because it works well. 6,0 magnitude is still the reference and we will keep it for all the other classification algorithms.	0,70	The accuracy improved again. But we observe that class 0 (earthquakes with Mag < 6.0) has better performance than Class 1 (earthquakes with Mag >= 6.0), as seen in the precision, recall, and f1-score.all, and f1-score.
Decision tree	Not better than the random forest classifier and took too much time to run. Not the best solution.	0,64	Again biased toward Class 0: Similar to Logistic Regression, the Random Forest is biased toward the majority class, as shown by higher recall and F1-score for Class 0.
K-Nearest Neighbors (KNN)	Easy to implement with sklearn. We used the 5 nearest neighbors to compare the distance and then classify if it was a magnitude over 6,0 or not.	0,67	Has an accuracy slightly lower than the decision tree but way easier to code especially if we use the library sklearn.
Support Vector Machine (SVM)	Keeping up with the classification problem, magnitude lower or over 6,0.	0,63	Accuracy very slightly under the previous ones, we can admit that they have a similar performance. Also easy to code because we used sklearn.
Hyperparameter tuning	Using this concept to try to improve our previous algorithms. The Random Forest Classifier algorithm will improve and apply hyperparameter tuning for the Random Forest Classifier algorithm to find the optimal value of n_neighbors and potentially tune other hyperparameters like weights and metric.	0,72	Using hyperparameter tuning allows us to have a higher accuracy than before. We still use sklearn to code, making it easier to implement.
SMOTE (Synthetic Minority Over-sampling Technique)	SMOTE did not improve the model significantly, it appears to have had little impact on the model's performance in this case. The accuracy, precision, recall, and F1-score for both classes (particularly Class 1) remain the same.		Not especially effective so we will abandon it for now. But it was a good thing to give it a try, in case it would have worked.

XGBoost	The XGBoost model has achieved a reasonably good accuracy (72%) on the dataset. This model excels in predicting class 0, with a recall of 92%. This suggests that the model can effectively capture the majority class.	0,72	One of the higher accuracy we had until now. But, the recall for class 1 is quite low (37%), which suggests the model misses many of the instances where the magnitude is ≥ 6.0. This indicates that the model might be biased toward predicting class 0, a common issue in imbalanced datasets.
LightGBM Regression	We tried a regression algorithm this time. The LightGBM Regression model shows moderate performance, with an RMSE of 0.43 and an R^2 score of 0.20, indicating that there is significant room for improvement. The model is not yet capturing the patterns in the data effectively. To enhance performance, consider further hyperparameter tuning, feature engineering, and addressing potential issues like underfitting.	RMSE = 0,43 R^2 = 0,20	RMSE and R^2 are really bad so we will not keep up with regression algorithms.
CatBoost classifier	The highest accuracy that we ever got. We got some issues at the beginning while installing the catboost but after that it was good. Easy to code because we only use algorithms from pandas, sklearn and catboost.	0,72	The CatBoost Classifier has a good accuracy of 72% overall, showing that it is effectively identifying the majority class (Class 0). It has a high recall for Class 0 (0.90), meaning it is good at identifying true positives for the majority class.
Deep Neural Network (DNN)	The DNN model is overfitting to the majority class (class 0) and fails to detect class 1, as evidenced by the precision, recall, and F1-score of 0 for class 1. Addressing class imbalance, fine-tuning the model, and applying strategies like SMOTE or class weighting would likely improve performance for class 1, helping the model achieve a better balance between precision and recall across both classes.	Crash	Did not work, but probably not the best option of algorithm for our project. Still a classifier algorithm but we have better options.
Gradient boosting classifier	The Gradient Boosting Classifier achieves an overall accuracy of 71%, with strong performance on the majority class (Class 0) and weak performance on the minority class (Class 1). The high recall for Class 0 and low recall for Class 1 indicate class imbalance issues, which are common across many models trained on imbalanced datasets. To improve performance, techniques like class balancing, hyperparameter tuning, and feature engineering are recommended.	0,71	We used SMOTE in this part, which helped us to get this quite high accuracy. Without SMOTE, the model performs better for the majority class but struggles to detect the minority class.

NACCACHE Lisa
NEJJARI Hiba
PRIEUR Leina
ON Milly

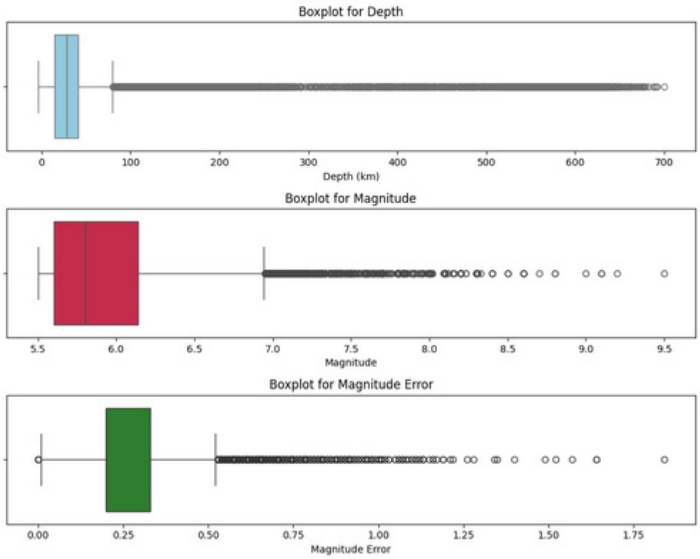


Decision tree classifier (took ~5 minutes to run)

In phase 3 of our project, we tried to upgrade some of our previous algorithms. Firstly, we decided to redo the data preprocessing. In stage 1, we noticed that some columns had many outliers, but we didn't take any action to address them, so we will handle them correctly now. Since we plan to use KNN, which is sensitive to outliers, we will handle the outliers first to ensure better results.

In stage 1, we used a boxplot, which is based on percentiles. In another course (Programming in Data Science), we learned about the z-score method, so we thought about using it here. However, it is less effective for heavily skewed or non-Gaussian data. As we saw in stage 1's visualisation, the 'Mag' and 'Depth' columns are right-skewed, so the z-score method isn't a good fit for our data. Therefore, we will stick with the boxplot method.

We can see that Depth (5168), Mag (1580), magError (1296) have a substantial number of outliers, so let's review them carefully. For the other columns, the percentage of outliers is very small (0.1%-2%) and doesn't significantly affect results, so we will keep them.



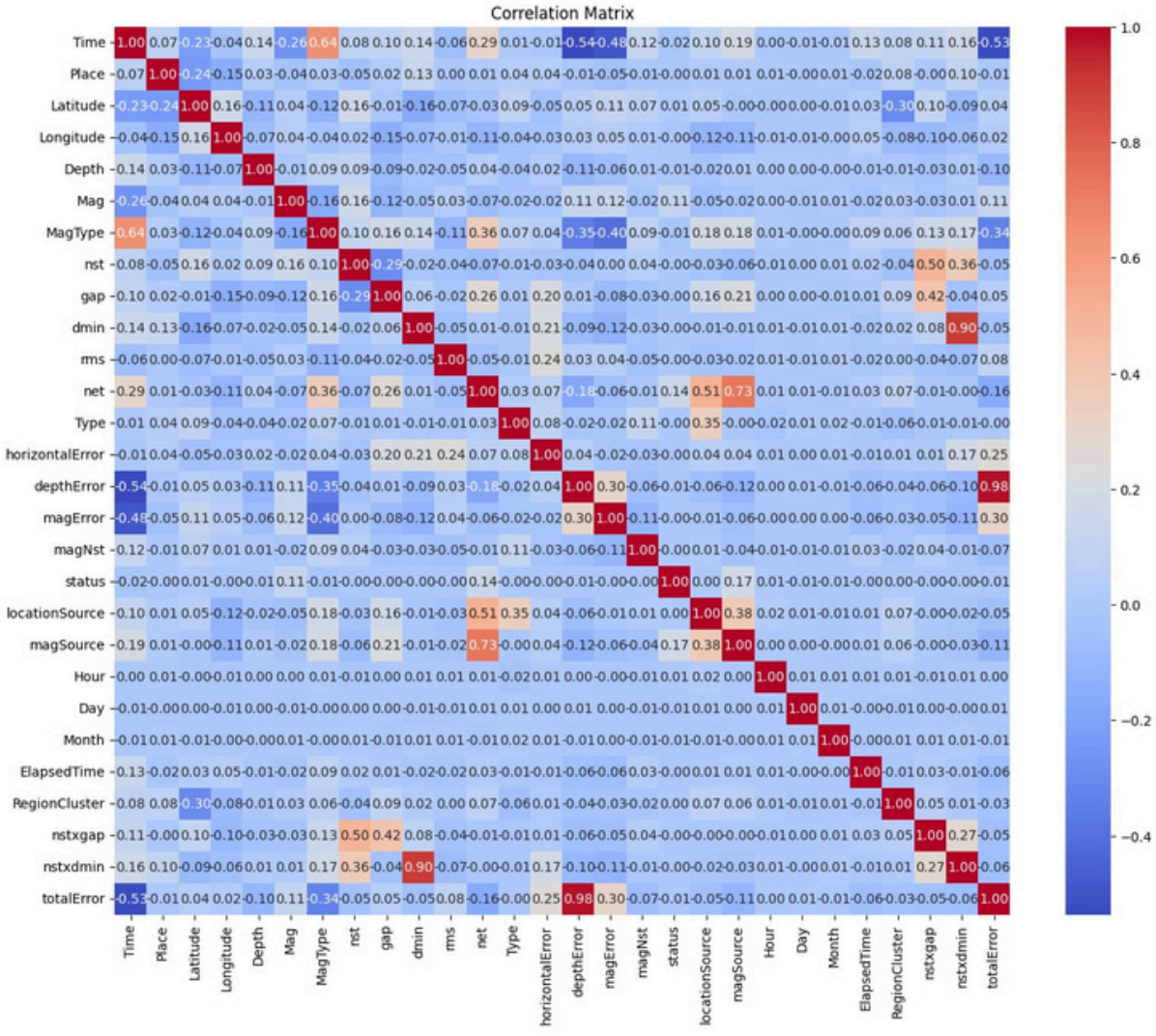
Boxplot of the magnitude before calculating the outliers

NACCACHE Lisa
NEJJARI Hiba
PRIEUR Leina
ON Milly

According to this website, the maximum depth is 700 km, but a minimum depth of -4 km is unusual, so we will remove these rows. For Mag the values should range between 0 and 10 based on the Richter scale, which confirms that our data is realistic. For magError, values are realistic and consistent with measurement uncertainty.

Then, we checked if there were any missing values and we used KNN for missing values filling. After that, we created new temporal, geological, seismicity-based and error-based features to amplify our dataset. We converted categorical values to numerical values, which will help us to handle them easier in the future. We also applied different conversions such as changing the given date in the dataset into a DateTime format, using dt. After all the modifications, we computed them into a new dataset.

In the next step, we calculated the correlation matrix but with more parameters than the first one that we previously did.

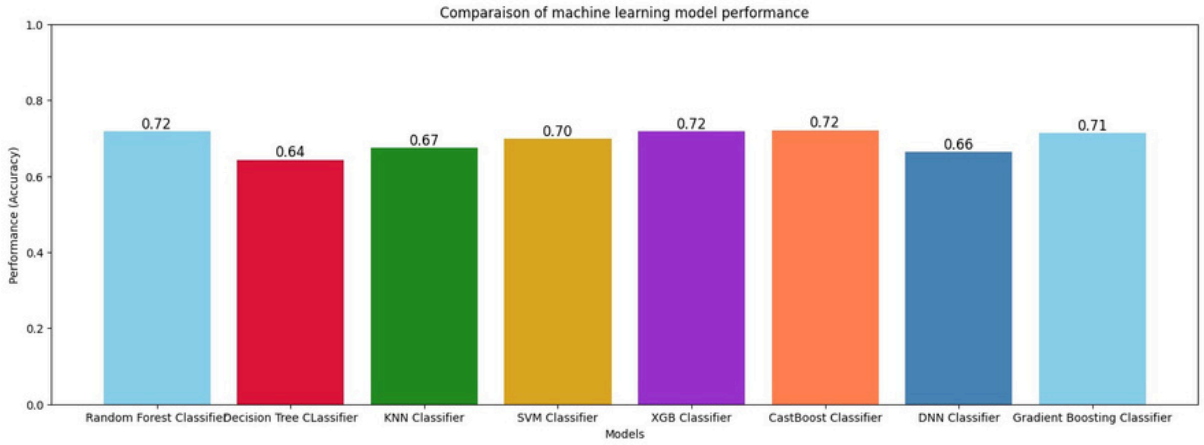


Second correlation matrix

To define the key variables, we selected our features based on their correlation strength and domain relevance. A threshold of approximately |0.1| or higher was used to include weaker but potentially relevant predictors (like magSource and magError). We also included important features like the latitude, longitude and depth because they provide

crucial spatial and contextual information that helps in understanding the characteristics and potential impact of earthquakes.

Using our new dataset after the second preprocessing, we reapplied our previous algorithms to compare which one has the higher accuracy. We made a plot to visualise it.



Plot comparing the accuracy of the classification models in stage 1 and 2

We can see that Random Forest, XGB Classifier, and CatBoost Classifier have the best performance. Therefore, in stage 3, we focused on these three models and worked on improving their performance. To improve our model, we did:

1. Hyperparametertuning:weusedtechniqueslikegridsearchtooptimizeparameters.
2. Classimbalance:weaddressedbiastowardthemajorityclassusingmethodslike threshold moving.
3. Ensemble Methods: we combined the models for better generalization.

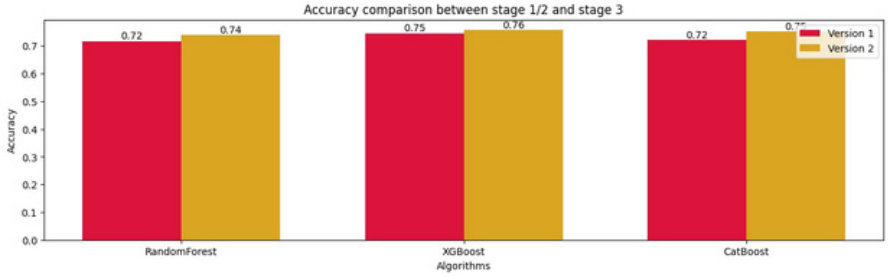
Talking about more technical aspects, during our search for a dataset, we came across a notebook on kaggle that used the same dataset as ours for predictions. The title claimed an accuracy of 1.0. Achieving 100% accuracy is quite shocking because it suggests that all predictions are perfectly correct, implying absolute certainty in every case. After carefully examining the suspicious code, we discovered that it used X instead of X_train, which, of course, guarantees 100% "accuracy" since the model is tested on the same data it was trained on. To verify, we tried running the same code but replaced X with X_train, and unsurprisingly, we got an accuracy much lower than what we achieved with the basic algorithms in our project.

For our deep learning algorithm, we have chosen to use the Neural Network built using TensorFlow and Keras. We have found a documentation about it on Google Scholar, called “Deep Learning with Python, Develop Deep Learning Models on Theano and TensorFlow Using Keras” by Jason Brownlee. This model is relevant for binary classification of earthquake magnitudes, particularly for categorizing them as "large" or "small" (here we are predicting whether the magnitude is greater than or equal to 6.0 or not). Neural networks are a good choice because they can capture complex non-linear relationships in our data.

The model's accuracy improved steadily over 10 epochs, reaching a final training accuracy of 72.1% and achieving 71.5% accuracy on the test set, showing consistent performance. To improve, we retrained the model with more epochs, but despite the longer execution time, the improvement was minimal. The final test accuracy increased slightly from

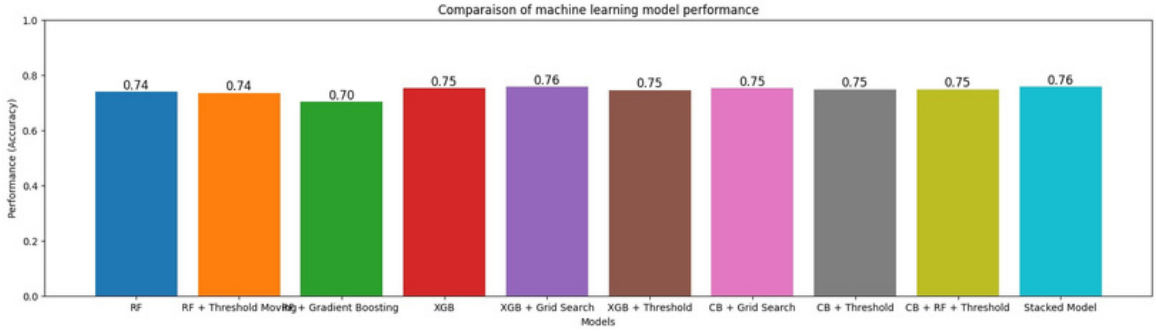
72.1% to 73.1%, which is good, but not the best one so far. It shows that deep learning algorithms don’t always mean that they are more efficient than the others.

And so, after doing all these predictions on our dataset, we made some visualisations to better understand the improvements.



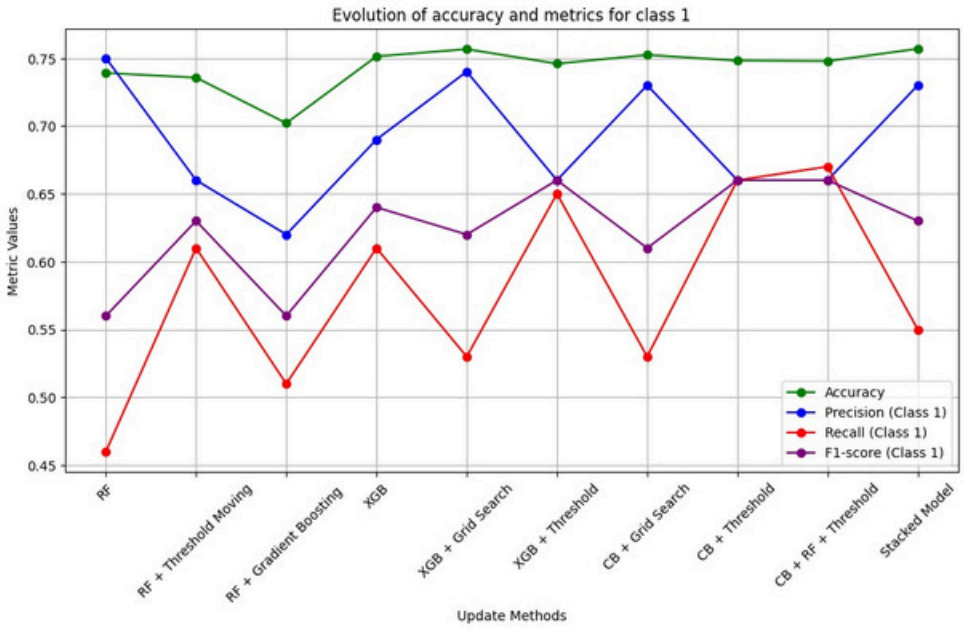
Plot comparing the accuracy between the different models in stage 1/2 and 3

Here we can see that the accuracy of all the algorithms improved a bit after the updated preprocessing method, new key variables, and hyperparameter tuning.



Plot comparing the accuracy of the different models in stage 3

Herewecanseethattheaccuracyofeachmodelconvergesaround0.76,whichis pretty good, but not revolutionary.



Plot showing the metrics of the different models applied in stage 3

Here we can see that the precision, recall, and f1-score for class 1 of the algorithms improved a bit after hyperparameter tuning, handling data imbalance, and combining algorithms.

On one hand, Grid Search helps improve the precision, but the performance of recall and f1-score decreases. On the other hand, threshold moving helps improve recall and f1-score, but the performance of the precision decreases. In the end, we concluded that the best model is CatBoost combined with Random Forest and threshold moving.

IV. Discussion and Conclusion

After doing all these predictions on the dataset we have chosen, we can conclude that we can predict if an earthquake's magnitude will be higher or lower than 6,0 on the Richter scale, with an accuracy of 75% using CatBoost combined with RandomForest and threshold moving. The recall and f1-score are still quite low, so it still needs improvement, but the performance has greatly improved from stage 2. Recall increased from 0.42 to 0.67, and f1-score increased from 0.52 to 0.66. With these results we hope that we could help the population to be more prepared about earthquakes in the future. This can include adapting architecture with specific materials, building lower towers, avoiding a specific location and helping people to move out from these dangerous zones.

Through this project, we have grown in coding, critical thinking, decision-making, organization, and AI knowledge. It was a truly interesting experience because we understood the purpose and goals of our work.