Prediction of building's collapse from effect of earthquakes, based on Gorkha Earthquake Dataset

Supervised binary classification algorithms comparison in Python

ABSTRACT

The purpose of this study is to predict if a building is subject to collapse by the effect of earthquakes, through machine learning algorithms, undertaking a comparison of their perform. This study approaches the problem as a supervised binary classifications case, using a dataset related to the Gorkha 7,8 magnitude earthquake, which happened in the district of Gandaki, Pradesh, Nepal in April 2015. The consequences of this earthquake were at a disaster dimension level, with around 9000 fatalities and millions of people who lost their houses, with circa $10 billion lost in infrastructures.

This dataset is one of the largest datasets registered after this kind of catastrophe, focused on building damage, with several files available that add information related to the financial, social, geographical and structural conditions of the area and the buildings involved. Although this dataset has a multivariable target as the grade of damage of the related buildings, this study has generated a customise dataset and target, establishing a binary variable for "Building collapse", and a dataset extracted from the different files and features available, selecting and generating which have been considered relevant for the study. These facts make this study very special, as even when this dataset has been used several times for machine learning approaches, the tailored target and features make this challenge unique as there are not previous studies with this approach.

To embrace this challenge, different algorithms has been applied, based on the state of the art of binary classification problems, such Extreme Gradient Boosting, as in [28], [13], [40], [15] and [33], Support Vector Machine, Decision Trees and Random Forest, and different combination of the dataset have been used to train the different models, as a result of feature engineering processes, obtaining different values of perform.

CONTENTS

INTRODUCTION

Earthquakes, according "Encyclopædia Britannica", is *"any sudden shaking of the ground caused by the passage of seismic waves through Earth's rocks"*. The Earth is conformed, in its most superficial layers, by the Earth's crust, a thick cover of rock that involves the whole planet. Sometimes, there is an accumulation of energy on at that level, and in some occasions, due to the fracture of masses of rock straining against one another, this leads into a fracture of the crust, releasing the energy accumulated and generating seismic waves, that travels along the crust, producing movement at an scale that could constitute a catastrophe, as the case of Gorkha earthquake.

The case related to the dataset in study, happened in the area of Kathmandu, in Nepal, generating the collapse of 600.000 structures in Kathmandu and surrounding towns, leaving almost 9000 deaths, 16800 injured, 2.8 million people lost their homes and, in total, around 8 million people were affected in some severe way.

Figure 1 [32]: Images from left up to bottom right: Ground fissures in the Trishuli dam reservoir, Settlement of the Araniko Highway, Bridge failure and building collapsed.

In the area of central Nepal, involving Kathmandu and Kathmandu Valley, was reported as a massive infrastructure and homes were destroyed, for several more than 600.000 houses

collapsing and almost 300.000 damaged. Other infrastructures, such schools, hospitals, public buildings, monasteries, etc. were affected by thousands, and not only in Nepal, but also in India, Bangladesh and Tibet areas. The destructive power of this earthquake was significative in comparison with other events, particularly dramatic since services such electricity and water supply were compromised as plenty of related infrastructures were destroyed as well. The fact that the roads were seriously affected and blocked, made even more difficult the rescue and logistic operations.

In relation with the case of this study, building collapse, the building typology in Nepal plays an important role for the content of the data. Most of the buildings in Nepal are constructed using adobe, brick or stone masonry joint with mud mortar or cement mortar, wooden structures are used, as well as reinforced concrete (RC). The case of residential buildings is, for small cities or villages, the use of wooden structures and adobe, but for the main cities and new developments it is used reinforced concrete in large buildings.

Following, the figure 2, showing the distribution of materials by percentage of buildings [37]:

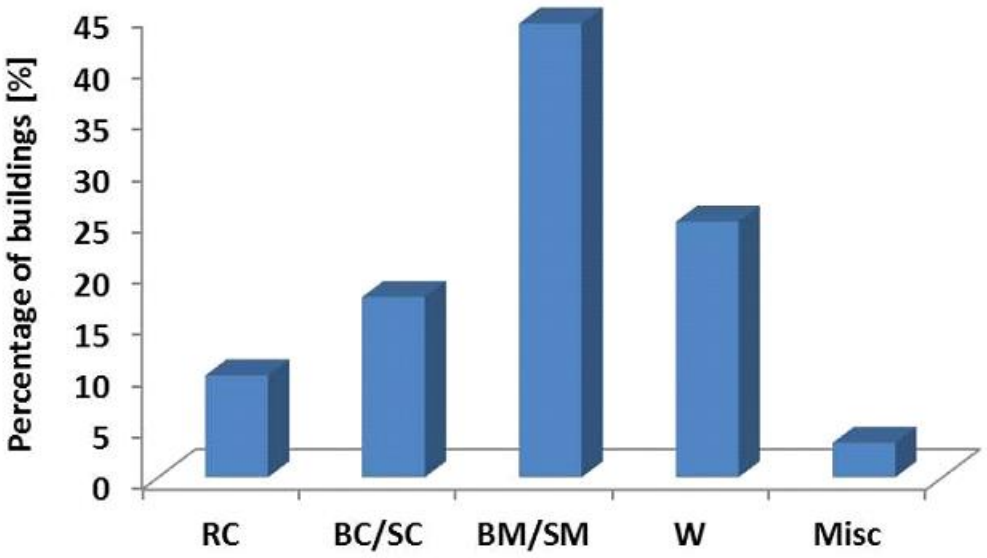

Fig.2: Materials of construction for buildings, by percentage of number of buildings.

In other hand, this dramatic catastrophe has left plenty of information to apply learn for future events and one of the largest datasets related to building damage by effect of earthquakes. This dataset has been used in several general studies, such [8], [34], [33] and [12], and in specific studies related to civil and geo-engineering and machine learning methodologies, such [42], [41], [43], [9], [13], [11], [25], [35], [16], [8], [31],[21], [17].
This has provided information to generate lesson learnt reports, guides for actions in future operations related to earthquake disasters, etc.

STATE OF THE ART

As mentioned in the Abstract of this report, this case of study has not been undertaken before, as the dataset and the targets has been tailored for this particular study. However, the use of the main dataset, the Gorkha or Nepal dataset, it is something very extended in the research field, as its size and information contained, involves a wide range of fields related to geoscience, seismology, structural design, socio-demography, etc.
In the field of Data Science, there are several studies that are based on the mentioned dataset, as per [13], [43], [31] and [35].

These publications use different algorithms and methodologies such, Linear Regression, Support Vector Regression, Gradient Boosting Regression, Random Forest Regression, Gradient Boosting Classification, and Random Forest Classification. Random Forest Regression shows good capabilities for damage prediction, according [8]. Other approaches focus on XGBoost, an Ensemble Learning technique, as [36], to overcome subjectiveness or limited features involved. Both algorithms will be tested in this study.

Regarding general binary classification algorithms publications, as per [28] and [15], XGBoost and Neural Network [22], [23], [29], [39], comparison and studies have been made, and SVM has been used for binary classifications handling imbalance datasets [24], [27], as per [38]. Also, Multilayer Perceptron Neural Networks are a typical approach for classification problems, as in [7].

In this study, all mentioned algorithms will be tested, as per the following pipeline figure, which phases will be explained in the coming sections.

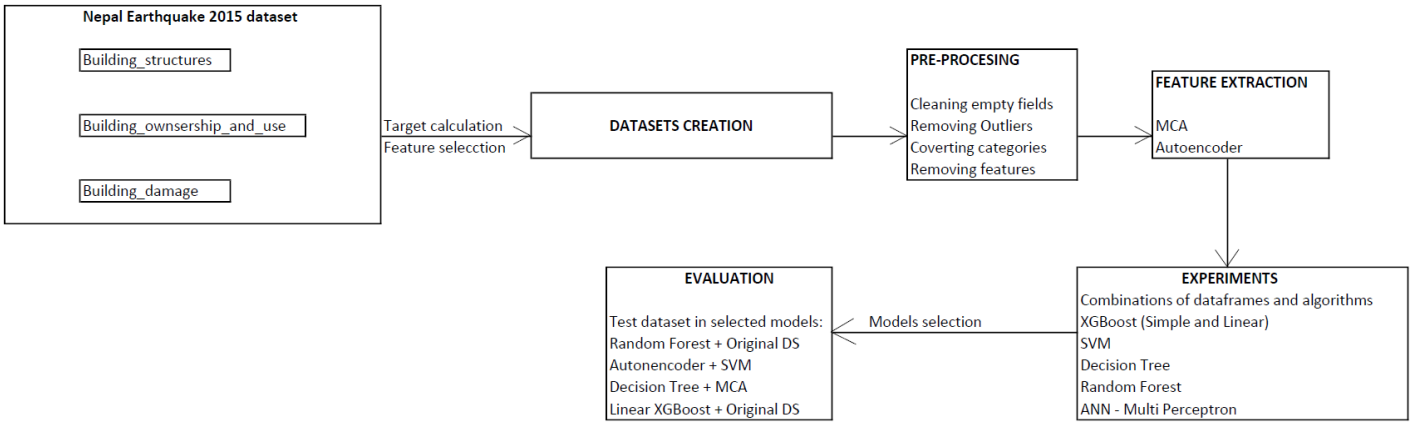

Fig. 3: Pipeline for the execution of the project.

GENERATION AND PRE-PROCESSING OF THE DATASET

The dataset used in this study is based, as mentioned, on Nepal or Gorkha earthquake dataset. This dataset is conformed from several files as follows:

- building_damage_assessment.csv
- building_ownership_and_use.csv
- building_structure.csv
- household_demographics.csv
- household_earthquake_impact.csv
- household_resources.csv
- individual_demographics.csv
- mapping.csv
- ward_vdcmun_district_name_mapping.csv

For the generation of the dataset for this study, only the files 'building_damage_assessment', 'building_ownership_and_use' and 'building_structure' have been used, extracting the features that were considered relevant for this approach.
The file 'building_damage_assessment' was used to establish the target feature, from the feature 'damage_overall_collapse', a multi variable categorical feature, with fuzzy values. This feature has been encoded, creating the target value 'Building_collapse', a binary variable numerical feature. The distribution of the values has been done according to the following table:

Damage_overall_assessment	Building_collapse (target feature)
'Moderate-Heavy'	1
'Severe-Extreme'	1
'Insignificant/light'	0
'None'	0

Fig.4: Criteria for generating target feature 'building_collapse'

The rest of the dataset has been create adding to the file 'building structure.csv', with already 15 features, the generated target feature 'Building_collapse', plus another 20 additional relevant features extracted from the file 'building_ownership_and_use'. In total, the original dataset, before its pre-processing phase, has 36 features, which 7 of them were categorical and the rest numerical, as float and integer variables.

There were 3 multivariable categorical features that were converted to numerical, as conceptually, were representing through those categories different ordinal levels of the features, based on the materials of each category. These features were 'land_surface_condition', converted into 'inclination' feature, 'roof_type', converted to 'roof_robustness' and 'position', converted to 'stability' feature. They were given ordinal values from 1 to 3 or 4, depending on the feature.
Another modification of the added features has been to convert the rest of the feature with categorical variables to numerical, with the idea of having a better interpretation in the phase of exploratory analysis and give independence to some categories that might have more influence on the prediction.

other_floor_type		
Categories	**New feature**	**value**
TImber/Bamboo-Mud	O_Timber	1
		0
'Not applicable	O_NA	1
		0
Timber-Planck	O_Planck	1
		0
RCC/RB/RBC	O_RCC	1
		0

Fig.5: Creation of new binary numerical features from multivariable categorical features

This process generated several new features, having a total number for this dataset of 53 features.

To finalise of setting up the dataset for this study, cleaning of instances with empty values, and elimination of outlier from the features 'age_building' and 'plinth_area_sq_ft', led to some more empty field, removing the instances involved. The following histograms for those 2 features show the distribution of the data along them, before and after removing the outlier values:

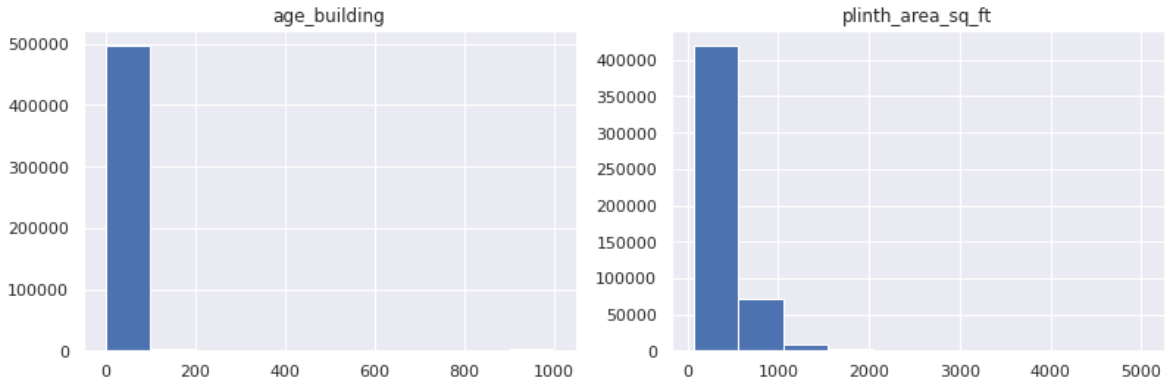

Fig. 6: Histograms of 'age_building' and 'plinth_area_sq_ft', before removing outlier values.

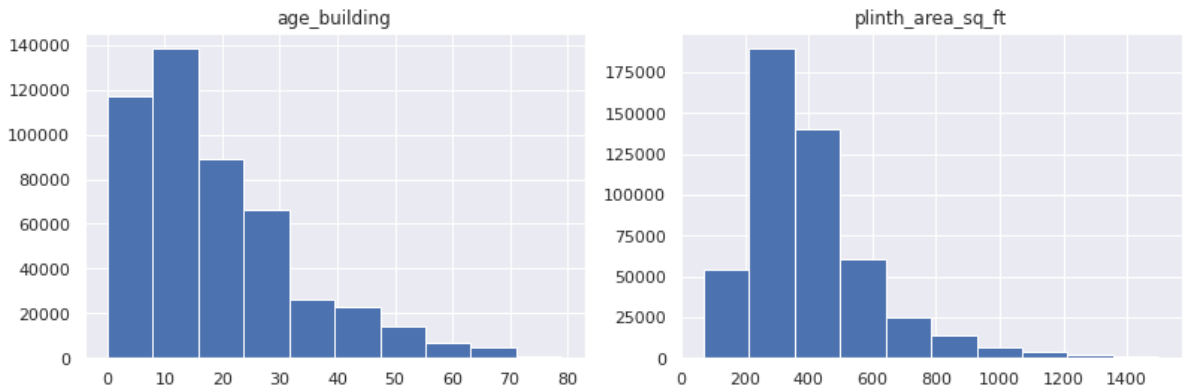

Fig. 7: Histograms of 'age_building' and 'plinth_area_sq_ft', after removing outlier values.

After all these processed, the final shape of the dataset is 487430 instances and 53 features, continuing to generate the final working datasets, splitting the main dataset into training and test, at a 60/40 proportion, and again, training and validation datasets, at an 80/20 proportion, generating dimensions such:

- Training dataset: 233966 instances x 53 features.
- Validation dataset: 58492 instances x 53 features.
- Test dataset: 194972 instances x 53 features.

EXPLORATORY ANALYSIS

Several checks have been done on the training data to understand the distribution of the information, focusing on the target data, and the features that show characteristics. As can be extracted from the figure 8 below, the dataset is imbalance, showing a rate between both binary target categories of 0.643, that correspond to 64,3% Positive and 35,7% Negative values.

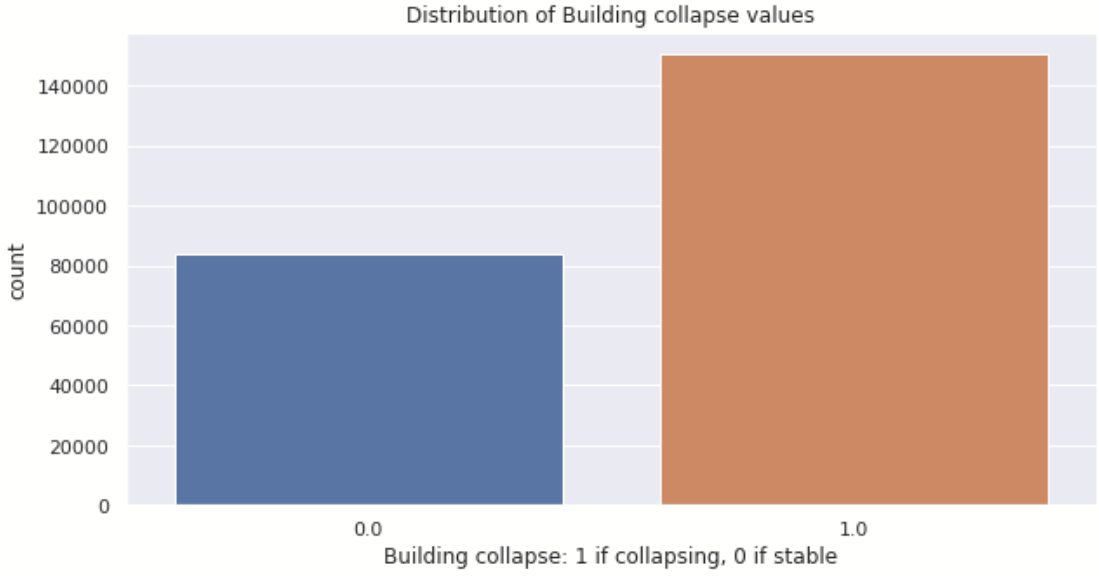

Fig.8: Histogram of the values of the target feature, ‘building_collapse’.

Continuing with the exploratory analysis, correlation matrix heatmaps of the features have been generate, as per the figure 9, showing that little correlation was present between features in the data, except for some categorical features, related to identifiers of buildings, districts, etc, that were a linear combination one from another and were removed leaving only the necessary ‘building_id’.

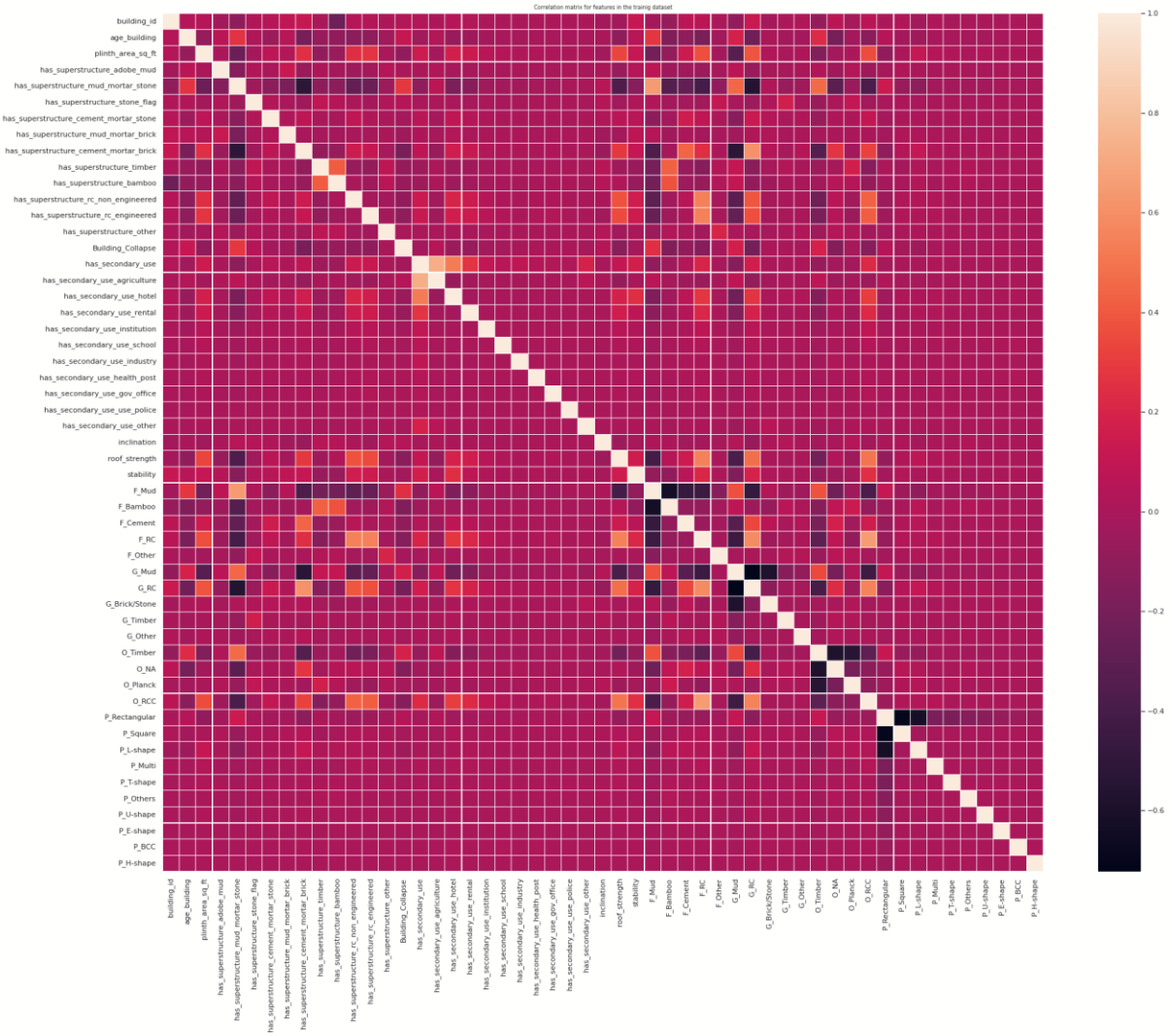

Fig. 9: Correlation matrix of the features heatmap.

This lack of correlation is more notorious in the table of correlation between features and the target, in figure 10, where the highest absolute values, reaching only between around 0.20 a 0.30, are related to only 4 features. These 4 mentioned features, named as 'F_Mud', ‘has_superstructure_cement_mortar_brick', 'has_superstructure_mud_mortar_stone' and 'G-RC' and are all of them binary variables, showing in the figure 11 the distribution of their values.

This lack of correlation between features and with target, forecast a lack of statistical information, necessary to deliver a valid prediction of the potential collapse of the buildings in study by the effect of the earthquake, according to the Nepal earthquake parameters.

Correlation with target feature 'building_collapse'

Positive correlation		Negative correlation	
Building_Collapse	1.000000	P_E-shape	-0.000784
has_superstructure_mud_mortar_stone	0.283371	has_superstructure_other	-0.000881
F_Mud	0.251209	has_secondary_use_gov_office	-0.002267
O_Timber	0.185164	G_Timber	-0.003338
G_Mud	0.166798	P_Multi	-0.004598
age_building	0.112472	P_BCC	-0.004888
has_secondary_use_agriculture	0.044219	P_U-shape	-0.007235
has_superstructure_stone_flag	0.042466	P_T-shape	-0.008166
P_Rectangular	0.038564	has_secondary_use_industry	-0.008261
inclination	0.038100	has_secondary_use_other	-0.009994
building_id	0.032272	P_Others	-0.011119
has_superstructure_adobe_mud	0.010422	has_secondary_use_school	-0.011151
G_Brick/Stone	0.006640	O_Planck	-0.012411
has_secondary_use_use_police	0.002761	P_Square	-0.013706
has_secondary_use_health_post	0.000802	G_Other	-0.014290
P_H-shape	0.000004	has_superstructure_cement_mortar_stone	-0.016163
		stability	-0.016866
		has_secondary_use_institution	-0.017606
		F_Other	-0.021076
		has_secondary_use	-0.027096
		P_L-shape	-0.037264
		has_superstructure_mud_mortar_brick	-0.040617
		has_secondary_use_hotel	-0.062293
		has_superstructure_bamboo	-0.065799
		has_secondary_use_rental	-0.067260
		has_superstructure_timber	-0.076879
		has_superstructure_rc_non_engineered	-0.091326
		F_Cement	-0.094469
		plinth_area_sq_ft	-0.096202
		has_superstructure_rc_engineered	-0.100274
		roof_strength	-0.127223
		O_RCC	-0.135038
		F_RC	-0.139894
		O_NA	-0.144291
		F_Bamboo	-0.160220
		has_superstructure_cement_mortar_brick	-0.195323
		G_RC	-0.210189

Fig.10: Correlation between features and target.

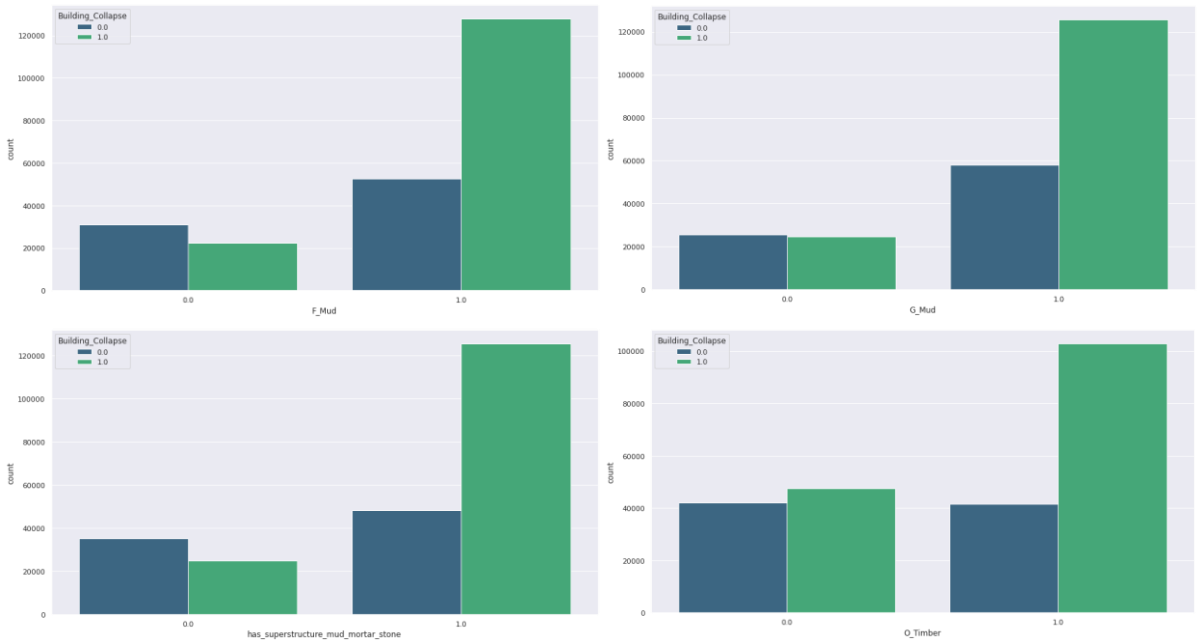

Fig.11: Values distribution of the features with more correlation with the target 'building_collapse': F_Mud, G_Mud, 'has_superstructure_mud_mortar_stone' and 'O_Timber'.

FEATURE ENGINEERING

Feature extraction.

Having a dataset with high dimensionality might be an issue when training the algorithms, due to the risk of overfitting, as per [6], [15] and [30]. For the reason, both manual and algorithm-based dimensionality reductions have been undertaken, generating different sets of data extract from the training, validation and test datasets, to be applied during the experiments phase. During this process, the features with numerical variables have set apart, in a different dataframe, and the rest of the features have been prepared for a multi correspondence analysis (MCA). This kind of dimensionality reduction acts only on categorical variables. For that reason, the left features in the dataset, after extracting the numerical features, have been converted into categorical variables. The decision of applying MCA dimensionality reduction is the fact that the features taking part of it, even being numerical variables, were binary variables, already acting as categories rather than as continue variables. Due that circumstance, Principal Component Analysis (PCA), that is one of the most common dimensionality reduction methodologies, has been replaced by MCA, again, due to the categorical nature of the features involved [20].

Therefore, a selection of 20 now categorical features, avoiding the ones with almost no statistical information, based on the lack of correlation with the target, have performed a Multi Correspondence Analysis, generating 2 new features, which are meant to gather the statistical information from the 20 features involved.

Moreover, dimensionality reduction technique Autoencoder has been generated, based on the training dataset of 53 features. This dimensionality reduction technique, is a neural network that has been set up with 5 hidden layers for this case, generating a dataset with 5 features, which are meant to collect the no linear statistical information of the original dataset. This methodology is an unsupervised learning method that encode the dataset and decoded again, trying to find the similarity between the results and the original data. It is meant to reduce this way the noise of the dataset. In this study, it will be applied to different algorithms and combined with other datasets.

Therefore, there are 4 different datasets from the training and validation data, that will be used to when training and validating different models, i.e., the original dataset, with 53 features, the autoencoder generated dataset, with 5 new features, based on the mentioned original dataset dimensionality reduction, and the MCA dataset, with the 2 new features generated from apply that dimensionality reduction technique. These different datasets will be used independently and in combination with each other, when training the models.

Feature scaling.

The next step carried out with the datasets has been the normalisation. The need of normalisation comes when the data of the features are in different scales, as it is this case, where most of the features are binary variables, with '0' or '1' values, but some numerical ones, as 'age_building', have range data from 0 to 80 or 'plinth_area_sq_ft', with range data from 0 to 1500 units. Normalisation is a technique that allows to establish a same data range for all features, speeding the training of the mode, [10], [14] and improving the results [18], [27], [19], [1], [42], and [26], in most of the cases, depending of the algorithm to train.

When the algorithm is Euclidean distance-based, such K-nearest neighbour, it will be strongly affected by normalisation, but in the case of Tree-based algorithm, it would affect less.

There are several types of normalisation techniques. In this study Min-Max Scaling. This normalisation technique changes the range of the data to a new range between 0 and 1. It has been applied, due to the type of algorithms used in this study that are distance-based, and the fact that all the features have already a range between 0 and 1, due to their binary nature. This technique is overly sensitive to the presence of outliers, but in this case, although there were a couple of features that had some, they have been already removed in the pre-processing phase.

The formula to calculate Min-Max Scaler is as follows:

$$X_{new} = \frac{X_i - \min(X)}{\max(x) - \min(X)}$$

Fig.12: Formula of Min-Max Scaling

Where X_{new} is the new value of the feature, X_i is the current value of the instance at a certain feature, max(x) is the maximum value of the instance for that feature and min(x) is the minimum value for that feature.

Below, can be seen the values of some notoriously affected features by the normalisation, before, figure 13, and after normalisation, figure 14.

	building_id	age_building	plinth_area_sq_ft	has_superstructure_adobe_mud	has_superstructure_mud_mortar_stone
count	5.007350e+05	500735.000000	500735.000000	500735.000000	500735.000000
mean	2.602952e+11	23.855329	415.632101	0.047175	0.740258
std	6.225410e+10	65.769190	242.432962	0.212012	0.438494
min	1.201010e+11	0.000000	70.000000	0.000000	0.000000
25%	2.135080e+11	8.000000	280.000000	0.000000	0.000000
50%	2.469030e+11	15.000000	360.000000	0.000000	1.000000
75%	3.108040e+11	26.000000	480.000000	0.000000	1.000000
max	3.667090e+11	999.000000	5000.000000	1.000000	1.000000

Fig.13: Describe command output before normalisation with Min-Max Scaling.

	building_id	age_building	plinth_area_sq_ft	has_superstructure_adobe_mud	has_superstructure_mud_mortar_stone
count	487430.000000	487430.000000	487430.000000	487430.000000	487430.000000
mean	0.567726	0.234791	0.234720	0.045974	0.742981
std	0.253162	0.181112	0.139982	0.209428	0.436991
min	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.378362	0.101266	0.146956	0.000000	0.000000
50%	0.513791	0.189873	0.202939	0.000000	1.000000
75%	0.772911	0.316456	0.286914	0.000000	1.000000
max	1.000000	1.000000	1.000000	1.000000	1.000000

Fig.14: Describe command output before normalisation with Min-Max Scaling.

EXPERIMENTS

For this study, a series of different algorithm have been trained, based on different combinations of the generated datasets. These algorithms are tree-based type some of them, such Decision Tree, Random Forest or XGBoost classifier, and linear models such Support Vector Machine (SVM), or Neural Network based models such Multilayer Perceptron. These algorithms have been trained using the original training dataset, with dimensions of almost 500.000 instances and 53 features, but as well the dataset created from applying dimensionality reduction of this dataset, linear type, using Multi Correspondence Analysis (MCA), and no linear with Autoencoder. Following the list of algorithms and dataset trained:

- Original features + Linear booster XGBoost Classifier
- MCA components + Simple XGBoost Classifier
- MCA + original features + Simple XGBoost Classifier
- MCA components + SVM classifier
- Autoencoder features + Simple XGBoost Classifier
- Autoencoder + SVM classifier
- Original features + ANN (Multilayer Perceptron - MLP) classifier
- Autoencoder features + ANN (Multilayer Perceptron - MLP) classifier
- Original features with Decision tree
- MCA with Decision tree
- MCA + original features with Decision tree
- Autoencoder features with Decision tree
- Original features with Random Forest
- Original features + MCA with Random Forest

The result of these training has been validated with the related validation dataset, and some evaluation metrics have been calculated to be able to compare the performance of each approach. From this comparison, where the time for fit and time for predict have been considered, the 4 best methodologies have been selected to test the models with the test dataset.

The report showing the dataset selected, the feature generation method, the chosen algorithm, the parameters applied and the result for train and validation datasets of the 4 principal considered evaluation metrics are in the following tables.

metric	algorithm	dataset	feature_sel	train	val
f1	Random Forest		Original features	0.852106	0.838213
f1	Random Forest	MCA	MCA + Original features	0.853209	0.837389
f1	Decision tree		Original features	0.848592	0.828840
f1	Decision tree	MCA	MCA + Original features	0.849463	0.826001
f1	Simple XGBoost		Original features	0.790872	0.789956
f1	XGBoost	MCA	MCA + Original features	0.790205	0.789420
f1	SVM	AutoEncoder	AutoEncoder	0.787506	0.786466
f1	ANN		Original features	0.787188	0.785877
f1	SVM	MCA	MCA	0.786078	0.785063
f1	ANN	AutoEncoder	AutoEncoder	0.784202	0.782707
f1	XGBoost	MCA	MCA	0.783119	0.782086
f1	XGBoost	AutoEncoder	AutoEncoder	0.782327	0.781001
f1	Linear XGBoost		Original features	0.782473	0.780146
f1	Decision tree	MCA	MCA	0.790915	0.774269
f1	Decision tree	AutoEncoder	AutoEncoder	0.811740	0.767126

Table 1: F1 metric evaluation report for validation data.

metric	algorithm	feature_gen	feature_sel	train	val
recall	Linear XGBoost		Original features	1.000000	1.000000
recall	SVM	AutoEncoder	AutoEncoder	0.881501	0.882031

metric	algorithm	feature_gen	feature_sel	train	val
recall	Decision tree	MCA	MCA	0.870514	0.881763
recall	Simple XGBoost		Original features	0.879379	0.880908
recall	SVM	MCA	MCA	0.879073	0.880079
recall	XGBoost	MCA	MCA + Original features	0.877737	0.879224
recall	ANN	AutoEncoder	AutoEncoder	0.866657	0.867542
recall	Random Forest		Original features	0.877358	0.865617
recall	Random Forest	MCA	MCA + Original features	0.878508	0.865136
recall	XGBoost	MCA	MCA	0.854606	0.860591
recall	XGBoost	AutoEncoder	AutoEncoder	0.857386	0.859335
recall	ANN		Original features	0.847330	0.848776
recall	Decision tree		Original features	0.864648	0.846022
recall	Decision tree	MCA	MCA + Original features	0.867967	0.844659
recall	Decision tree	AutoEncoder	AutoEncoder	0.853908	0.810335

Table 2: Recall metric evaluation report for validation data.

metric	algorithm	feature_gen	feature_sel	train	val
accuracy	Random Forest		Original features	0.804271	0.786296
accuracy	Random Forest	MCA	MCA + Original features	0.805728	0.785116
accuracy	Decision tree		Original features	0.801706	0.776534
accuracy	Decision tree	MCA	MCA + Original features	0.802292	0.772413
accuracy	ANN		Original features	0.705564	0.704199
accuracy	Simple XGBoost		Original features	0.701115	0.700403
accuracy	XGBoost	MCA	MCA + Original features	0.700469	0.700010
accuracy	SVM	AutoEncoder	AutoEncoder	0.694272	0.693685
accuracy	XGBoost	MCA	MCA	0.695785	0.693291
accuracy	ANN	AutoEncoder	AutoEncoder	0.693460	0.691941

metric	algorithm	feature_gen	feature_sel	train	val
accuracy	SVM	MCA	MCA	0.692507	0.691804
accuracy	XGBoost	AutoEncoder	AutoEncoder	0.693370	0.691787
accuracy	Decision tree	AutoEncoder	AutoEncoder	0.745450	0.685359
accuracy	Decision tree	MCA	MCA	0.704205	0.671186
accuracy	Linear XGBoost		Original features	0.642675	0.639540

Table 3: Accuracy metric evaluation report for validation data.

metric	algorithm	feature_gen	feature_sel	train	val
type_ii_err	Linear XGBoost		Original features	1.000000	1.000000
type_ii_err	Decision tree	MCA	MCA	0.594914	0.702428
type_ii_err	SVM	MCA	MCA	0.643047	0.642241
type_ii_err	SVM	AutoEncoder	AutoEncoder	0.642473	0.640486
type_ii_err	Simple XGBoost		Original features	0.619507	0.619854
type_ii_err	ANN	AutoEncoder	AutoEncoder	0.618047	0.619617
type_ii_err	XGBoost	MCA	MCA + Original features	0.618358	0.617957
type_ii_err	XGBoost	AutoEncoder	AutoEncoder	0.601624	0.605483
type_ii_err	XGBoost	MCA	MCA	0.589866	0.603538
type_ii_err	ANN		Original features	0.549413	0.552315
type_ii_err	Decision tree	AutoEncoder	AutoEncoder	0.449618	0.536378
type_ii_err	Random Forest	MCA	MCA + Original features	0.325172	0.356858
type_ii_err	Decision tree	MCA	MCA + Original features	0.315830	0.355767
type_ii_err	Random Forest		Original features	0.327181	0.354439
type_ii_err	Decision tree		Original features	0.311500	0.346756

Table 4: Type 2 Error metric evaluation report for validation data.

From these tables will be selected the 4 algorithms that performs best, to be trained using the test dataset, and to study their behave.

EVALUATION METRICS

A set of evaluation metrics have been applied to the training models to compare their performance. All of them are based on confusion matrix, which is the combination of True/False Positive/Negative actual values and predictions, as per the figure 13. Different combinations and formulas based on these values conforms the evaluation metrics used in this study.

They type of evaluation metric chosen depends on the dominion of the study, being sometimes more important to avoid False negative than to be able to predict with precision. In the case of this study, that its aim is to predict the building that would collapse by the effect of an earthquake event, it is more important to assure that if a building is predicted to not collapse, be accurate, rather than being able to predict accurately the buildings that will not collapse, as a mistake in the first prediction would lead to a catastrophe, and a mistake in the second case, would not have that dramatic consequences.

For this reason, even when this study has calculated for the trained algorithms a set of 6 different evaluation metrics, the selection of the best methodology will be based on the metrics with high capabilities to minimise false negative cases, such Type II Error, Recall or Sensitivity and F1, having into account as well the accuracy, as an overall metric.

CONFUSION MATRIX

	Actually Positive (1)	Actually Negative (0)
Predicted Positive (1)	True Positive (TP)	False Positive (FP)
Predicted Negative (0)	False Negative (FN)	True Negative (TN)

Fig. 15: Confusion matrix

According to this metrics, following the results of the validation of the algorithms, the combination with better metric values as follows:

1. Original features training Linear XGBoost classifier.

2. MCA generated dataset training Decision Tree.
3. Autoencoder generated dataset with SVM classifier.
4. Original features training Random Forest.

These selected algorithms are Tree-based, except from SVM that is linear classification, and the involved datasets involved the three different approaches, the original dataset, with 53 features, mainly conformed by binary variables, apart from 'age_building' and 'plinth_area_sq_ft', MCA generated dataset, a dimensionality reduction product with only 2 dimensions, with a linear approach, and Autoencoder generated dataset, with 5 features, with a neuro network approach.

The values obtained from applying these algorithms to the test dataset are as follows:

MODEL		COST		EVALUATION METRICS			
Dataset	Algorithm	Time fit	Time process	Accuracy	Recall	Type 2 Error	F1
Original	Random Forest	134.46713	0.00009	0.784887	0.863520	0.356924	0.837787
Autoencoder	SVM	2.26628	0.00007	0.695813	0.882767	0.641350	0.788752
MCA	Decision Tree	0.13616	8.86917	0.688950	0.895141	0.682905	0.787350
Original	Linear XGBoost	1.01654	7.93934	0.643297	1.000000	1.000000	0.782935

Fig.16: Test dataset model report.

CONCLUSION

According to the previous table, show that the model with the best overall metrics, following the accuracy value, would be Random Forest, trained using the original dataset generated in this study, but having into account the cost of the model, that is much higher than the rest, with a fitting cost of almost 200% higher than the average of the other 3 models. Moreover, the value of the Type 2 Error metric is the lowest and the third in the case of recall metric. All these facts lead to refuse this solution. The case of Linear XGBoost, with and average cost, the lowest accuracy and suspicious values for recall and type 2 error makes it not the best option as well. The rest 2 models show similar performance with particularity that the SVM with autoencoder, have much better process time, being this option the most reasonable, considering that none of them are ideal.

The reason of the low perform of these models could be the particularity of the dataset, mainly made of binary variables with exceptionally low correlation between features. The fact that it is an original dataset, with tailored target that has never used in the past, with no previous studies with the same approach, has led as well in a lack of references or examples to rely on when choosing the methodologies to apply. In any case, it has been a challenging

exercise and leaves the path open to further improvement, which could be reach by adding additional features from other files of the data, undertaking an extended study in a bigger scale.

REFERENCES

[1] A. B. A. Graf, A. J. Smola and S. Borer, "Classification in a normalized feature space using support vector machines," in IEEE Transactions on Neural Networks, vol. 14, no. 3, pp. 597-605, May 2003, doi: 10.1109/TNN.2003.811708.

[2] A. Kumaraswamy, B. N. Reddy and R. Kolla, "Richter's Predictor: Modelling Earthquake Damage Using Multi-class Classification Models," 2020 Third International Conference on Advances in Electronics, Computers and Communications (ICAECC), 2020, pp. 1-5, doi: 10.1109/ICAECC50550.2020.9339484.

[3] [Bach, 2014] Francis Bach. Breaking the curse of dimensionality with convex neural networks. rXiv:1412.8690, 2014.

[4] Bandita Sijapati, et. al. “Migration and Resilience : Experiences from Nepal’s 2015 Earthquake”, Center for the study of labour and mobility.

[5] Ben Ramalingam and David Sanderson (2015), “Nepal Eartquake Response: Lessons for operational agencies”.

[6] Bo Liu, Ying Wei, Yu Zhang, Qiang Yang, "Deep Neural Networks for High Dimension, Low Sample Size Data", Hong Kong University of Science and Technology, Hong Kong, Twenty-Sixth International Joint Conference on Artificial Intelligence (IJCAI-17).

[7] Cristiano L. Castro and Antônio P. Braga “Novel Cost-Sensitive Approach to Improve the Multilayer Perceptron Performance on Imbalanced Data”. IEEE Transactions on neural networks and learning systems, Vol. 24, no. 6. June 2013.

[8] Ghimire S, Guéguen P, Giffard-Roisin S, Schorlemmer D. Testing machine learning models for seismic damage prediction at a regional scale using building-damage dataset compiled after the 2015 Gorkha Nepal earthquake. Earthquake Spectra. 2022. doi:10.1177/87552930221106495.

[9] G.Katsuichiro, K.Takashi, P.Rama, G. Chiaro, K. ToshihikoK.Sharma, S.Wilkinson. "The 2015 Gorkha Nepal earthquake: insights from earthquake damage survey", 22 June 2015. doi.org/10.3389/fbuil.2015.00008.

[10] Grus, Joel (2015). Data Science from Scratch. Sebastopol, CA: O'Reilly. pp. 99, 100. ISBN 978-1-491-90142-7.

[11] Hemchandra Chaulagain Hugo Rodrigues Enrico Spacone H. Varum(2015), “Seismic response of current RC buildings in Kathmandu Valley”.

[12] “Humanitarian crisis after Nepal earthquakes 2015”, World Health Organization (2015).

[13] J. Parashar, Sumiti, M. Rai, “Breast cancer images classification by clustering of ROI and mapping of features by CNN with XGBOOST learning” Maharishi Markandeshwar, Ambala 133207, Haryana, India, September 2020.

[14] Juszczak, P.; D. M. J. Tax; R. P. W. Dui (2002). "Feature scaling in support vector data descriptions". Proc. 8th Annu. Conf. Adv. School Comput. Imaging: 25–30. CiteSeerX 10.1.1.100.2524.

[15] J. Wu, Y. Li, Y. Ma, “Comparison of XGBoost and the Neural Network model on the class-balanced datasets”, 2021 IEEE 3rd International Conference on Frontiers Technology of Information and Computer (ICFTIC), 978-1-6654-0605-5/21/$31.00 ©2021 IEEE | doi:10.1109 / ICFTIC 54370.2021.9647373.

[16] Kurashimo, E., Sato, H., Sakai, S., Hirata, N., Gajurel, A. P., Adhikari, D. P., et al. (2019). The 2015 Gorkha earthquake: Earthquake reflection imaging of the source fault and connecting seismic structure with fault slip behavior. Geophysical Research Letters, 46, 3206– 3215. doi:/10.1029/2018GL081197.

[17] Lautour, Oliver & Omenzetter, Piotr. (2009). Prediction of seismic-induced structural damage using artificial neural networks. Engineering Structures. 31. 600-606. 10.1016/j.engstruct.2008.11.010.

[18] M. Azizjon, A. Jumabek and W. Kim, "1D CNN based network intrusion detection with normalization on imbalanced data," 2020 International Conference on Artificial Intelligence in Information and Communication (ICAIIC), 2020, pp. 218-224, doi: 10.1109/ICAIIC48513.2020.9064976.

[19] Meliboev Azizjon, Alikhanov Jumabek, Wooseong Kim. "1D CNN based network intrusion detection with normalization on imbalanced data".

[20] Nenadic, O. and Greenacre, M. (2007). Correspondence Analysis in R, with twoand three-dimensional graphics: The ca package. In Journal of Statistical Software, 20 (3): 1–13.

[21] Ömer Aydan and Resat Ulusay (2015), “A Quick Report On The 2015 Gorkha(Nepal) Eartquake And Its Geo-engineering Aspects”

[22] O. TURKSOY “HVAC Occupancy Detection with ML and DL Methods” , available at https://www.kaggle.com/code/turksoyomer/hvac-occupancy-detection-with-ml-and-dl-methods

[23] R. Alejo, V. García, J. Sotoca, R. A. Mollineda, and J. Sánchez, “Improving the performance of the RBF neural networks trained with imbalanced samples in Computational and Ambient

Intelligence" (Lecture Notes in Computer Science), vol. 4507. New York, USA: Springer-Verlag, 2007, pp. 162, 169.

[24] R. Dabare, K. Wai Wong, P. Koutsakis, M. Fairuz Shiratuddin. "A Study of the Effect of Dropout on Imbalanced Data Classification using Deep Neural Networks". Journal of Multidisciplinary Engineering Science and Technology (JMEST) ISSN: 2458-9403 Vol. 5 Issue 10, October – 2018

[25] Revathy M. Parameswaran, Thulasiraman Natarajan, Kusala Rajendran, C.P. Rajendran, Rishav Mallick, Matthew Wood, Harish C. Lekhak (2015), "Seismotectonics of the April–May 2015 Nepal earthquakes: An assessment based on the aftershock patterns, surface effects and deformational characteristics" .

[26] S.Gopal Krishna Patro, Kishore Kumar sahu. "Normalization: A Preprocessing Stage",Research Scholar, Department of CSE & IT, VSSUT, Burla, Odisha, India. 19 Mar 2015. ArXiv:1503.06462.

[27] S. Ioffe and C. Szegedy. (Mar. 2015). "Batch normalization: Accelerating deep network training by reducing internal covariate shift." [Online]. Available: https://arxiv.org/abs/1502.03167.

[28] S. M. H. Mahmud, W. Chen, H. Jaahan, Y. Liu, N. I. Sujan, S. Ahmed, "iDTi-CSsmoteB: Identification of Drug_Target Interaction Based on Drug Chemical Structure and Protein Sequence Using XGBoost With Over-Sampling Technique SMOTE", April 2019 doi: 10.1109/access 2019.2910277.

[29] S. Ranjit "Occupancy Detection using Machine Learning", Cochin University of Science and Technology, Kerala, India. IJRECE VOL. 6 ISSUE 2 APR.-JUNE 2018 ISSN: 2393-9028 (print) | ISSN: 2348-2281 (online).

[30] [Srivastava et al., 2014] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. Journal of Machine Learning Research, 15(1):1929–1958, 2014.

[31] Sujith Mangalathu, Han Sun, Chukwuebuka C. Nweke, Zhengxiang Yi and Henry V. Burton. "Classifying earthquake damage to buildings using machine learning". Earthquake Spectra (2020),36(1): 183. doi.:/10.1177/8755293019878137.

[32] Swathi.S, K.Venkataramana. "The Nepal earthquake 2015: A case of study". 5th International Engineering Symposium - IES 2016. March 2-4, 2016, Kumamoto University, Japan.

[33] S. Zhao, D. Zeng, W. Wanga, X. Chen, Z. Zhang, F. Xu, X. Maob, X. Liu, "Mutation grey wolf elite PSO balanced XGBoost for radar emitter individual identification based on measured signals", Journal homepage: www.elsevier.com/locate/ Measurement 159 (2020) 107777, March 2020.

[34] T.G. Sitharam, J.S. Vinod (2015), "Nepal Earthquake of April 25, 2015", International Journal of Geotechnical Earthquake Engineering, 6(1),81-90, January-June 2015.

[35] Thapa DR, Tao X, Fan F, Tao Z. Aftershock analysis of the 2015 Gorkha-Dolakha (Central Nepal) earthquake doublet. Heliyon. 2018 Jul 3;4(7):e00678. doi: 10.1016/j.heliyon.2018.e00678. PMID: 29998197; PMCID: PMC6037884.

[36] Toutiaee, Mohammadhossein. (2021). “Occupancy Detection in Room Using Sensor Data”.

[37] Upadhyay, Kamlesh & Kaur, Prabhjot & Prasad, Svav. (2021). “State of the Art on Data level methods to address Class Imbalance Problem in Binary Classification.” 8. 975-903.

[38] Wah, Y.B.; Rahman, H.A.A.; He, H.; Bulgiba, A. “Handling imbalanced dataset using SVM and KNN approach”. AIP Conf. Proc. 2016, 1750, 020023.

[39] Weiyi Chen, Limao Zhang. "Building vulnerability assessment in seismic areas using ensemble learning: A Nepal case study", Journal of Cleaner Production,Volume 350,2022,131418,ISSN 0959-6526,doi.org/10.1016/j.jclepro.2022.131418.

[40] X. Y. Liew, N. Hameed, J. Clos, “An investigation of XGBoost-based algorithm for breast cancer classification”, University of Nottingham, Jubilee Campus, Wollaton Road, Nottingham, NG8 1BB, United Kingdom, 2021.

[41] Yamazaki, F. & Liu, Wen & Bahri, Rendy & Sasagawa, Tadashi. (2016). Damage extraction of buildings in the 2015 Gorkha, Nepal earthquake from high-resolution SAR data. 98772K. 10.1117/12.2223306.

[42] Y. Wang and B. -G. Hu, "Derivations of Normalized Mutual Information in Binary Classifications," *2009 Sixth International Conference on Fuzzy Systems and Knowledge Discovery*, 2009, pp. 155-163, doi: 10.1109/FSKD.2009.342.

[43] 25 April 2015 Gorkha Earthquake Disaster Risk Reduction Situation Report”, DRR sitrep 2015-002 – June 2, 2015, UNISDR The United Nations Office for Disaster Risk Reduction.

Appendix 2: CODE

```
!pip install prince #for the use of MCA, of not already installed.

#IMPORTING LIBRARIES
import os #To manage files
import pandas as pd #to create panda dataframes
from google.colab import drive #to connect with Google Drive
import matplotlib.pyplot as plt #to plot graphs
import seaborn as sns; sns.set() #to plot graphs
from sklearn.model_selection import train_test_split # to split the dataset into train, validation and test datasets.
import matplotlib.ticker as mtick # to generate graphs
import numpy as np # linear algebra
from mpl_toolkits.mplot3d import Axes3D #to generate scatterplots
from sklearn.decomposition import PCA #to calculate Principal Component Analysis - PCA
import prince #to use multiple correspondece analysis - MCA
import plotly.graph_objs as go # for graphics
from plotly import tools #for graphics
import xgboost as xgb #to use XGBoost
from sklearn.metrics import accuracy_score, f1_score, roc_auc_score, confusion_matrix, recall_score, matthews_corrcoef #to use evaluation metrics
import tensorflow as tf #to develop Autoencoder
from tensorflow import keras #to develop Autoencoder
from sklearn.neural_network import MLPClassifier #to train ANN - MLP
from sklearn.tree import DecisionTreeClassifier #to train Decision Tree
import time #to check the processing time
from sklearn.svm import LinearSVC #to operate with linear SVM
from sklearn.ensemble import RandomForestClassifier #to train Random Forest model.

# %matplotlib inline

#function to create dataframe for Autoencoder
def create_new_features_dataframe(feature_numpy_array, column_labbel, norm=False):
  if norm:
    input_ = normalize(feature_numpy_array)
  else:
    input_ = feature_numpy_array

  df = pd.DataFrame(input_)
  df.columns = ["{}_{}".format(column_labbel, idx) for idx, _ in enumerate(df.columns)]

  return df

#Function to plot graphs in browser.
def configure_plotly_browser_state():
  import IPython
  display(IPython.core.display.HTML('''
```

```
    <script src='/static/components/requirejs/require.js'></script>
    <script>
      requirejs.config({
        paths: {
          base: '/static/base',
          plotly: 'https://cdn.plot.ly/plotly-latest.min.js?noext',
        },
      });
    </script>
    '''))
```

#Setting up colab to connect with Google Drive

```
drive.mount('/content/drive')
```

#Setting up colab to deisplay all rows.

```
pd.set_option('display.max_rows', None)
pd.set_option('display.max_columns', None)
```

"""Creation of the dataset.**"""**

#Reading data files

```
data = pd.read_csv('./data.csv')
b_damage = pd.read_csv('./b_damage.csv')
b_use = pd.read_csv('./b_use.csv')
```

#Creating 'collapse' feauture from b_damage, that will be the target.

```
for i in b_damage:
  b_damage.loc[b_damage['damage_overall_collapse'] == 'Moderate-Heavy', 'collapse'] = 1
  b_damage.loc[b_damage['damage_overall_collapse'] == 'Severe-Extreme', 'collapse'] = 1
  b_damage.loc[b_damage['damage_overall_collapse'] == 'Insignificant/light', 'collapse'] = 0
  b_damage.loc[b_damage['damage_overall_collapse'] == 'None', 'collapse'] = 0

b_damage['collapse'].describe()
```

#Adding 'collapse' feature from b_damage dataframe to main dataframe 'data', as the target feature.

```
data['Building_Collapse']=b_damage['collapse']
```

#Cheking b_use data.

```
b_use.info()
```

#Removing repeated and non-related features from b_use dataftrame, to add to whole dataset to 'data'.

```
b_use.drop(columns=['district_id', 'vdcmun_id', 'ward_id', 'legal_ownership_status', 'count_families'],
inplace=True)
```

#Adding b_use dataframe left features to main dataframe 'data'.

```
data = data.merge(b_use, on='building_id')
```

```
"""**Preprocessing of the dataset.**"""

#Removing instances with empty values in target feature.
data.drop(data[data.Building_Collapse.isnull()].index, inplace=True)

#Removing non-related features for this study.
data.drop(columns=['count_floors_pre_eq', 'count_floors_post_eq', 'height_ft_pre_eq',
'height_ft_post_eq', 'condition_post_eq', 'damage_grade', 'technical_solution_proposed'],  inplace=True)

#Looking for instances with empty values.
data_temp = data.isnull().sum().reset_index(name='count')
display(data_temp[data_temp['count'] > 0])

#Removing instances with empty values.
data.dropna(inplace=True)

#Checking if there are duplicate instances.
sum(data.duplicated())

data.shape

data.info()

#Heatmap of the correlation matrix
plt.figure(figsize=(30,24))
plt.title('Correlation matrix of features', fontdict={'fontsize':10})
ax = sns.heatmap(data.corr(), annot=False, linewidths=0.1)

#Removing
data.drop(columns=['district_id', 'vdcmun_id', 'ward_id'],  inplace=True)

#Converting the ID of the every instance 'Building_id' into a categorical variable.
data['building_id'] = data.building_id.astype('object')

data.info()

data.shape

"""Converting categorical variables into numerical variables."""

data['land_surface_condition'].value_counts()
#This is categorical ordinal data.

#Converting 'land_surface-condition' from 'object' to numerical values.
for i in data:
  data.loc[data['land_surface_condition'] == 'Flat', 'inclination'] = 1
  data.loc[data['land_surface_condition'] == 'Moderate slope', 'inclination'] = 2
  data.loc[data['land_surface_condition'] == 'Steep slope', 'inclination'] = 3
```

```
data.drop(columns=['land_surface_condition'],  inplace=True)

data['roof_type'].value_counts() #This could be considered categorical ordinal data, as it can be understood as 'roof robustness', as the hardest material, the more robust the roof could be.

#Converting 'roof_type' from 'object' to numerical values.
for i in data:
  data.loc[data['roof_type'] == 'Bamboo/Timber-Light roof', 'roof_strength'] = 1
  data.loc[data['roof_type'] == 'Bamboo/Timber-Heavy roof', 'roof_strength'] = 2
  data.loc[data['roof_type'] == 'RCC/RB/RBC', 'roof_strength'] = 3

data.drop(columns=['roof_type'],  inplace=True)

data['position'].value_counts()
#This could be considered categorical ordinal data, as it can be understood as 'stability', that the more attached to other structures, the more stable.

#Converting 'roof_type' from 'object' to numerical values.
for i in data:
  data.loc[data['position'] == 'Not attached', 'stability'] = 1
  data.loc[data['position'] == 'Attached-1 side', 'stability'] = 2
  data.loc[data['position'] == 'Attached-2 side', 'stability'] = 3
  data.loc[data['position'] == 'Attached-3 side', 'stability'] = 4

data.drop(columns=['position'],  inplace=True)

data['foundation_type'].value_counts()
#This is categorical nominal data.

#Converting 'foundation_type' from 'object' to numerical values.
for i in data:
  data.loc[data['foundation_type'] == 'Mud mortar-Stone/Brick', 'F_Mud'] = int(1)
  data.loc[data['foundation_type'] != 'Mud mortar-Stone/Brick', 'F_Mud'] = int(0)
  data.loc[data['foundation_type'] == 'Bamboo/Timber', 'F_Bamboo'] = 1
  data.loc[data['foundation_type'] != 'Bamboo/Timber', 'F_Bamboo'] = 0
  data.loc[data['foundation_type'] == 'Cement-Stone/Brick', 'F_Cement'] = 1
  data.loc[data['foundation_type'] != 'Cement-Stone/Brick', 'F_Cement'] = 0
  data.loc[data['foundation_type'] == 'RC', 'F_RC'] = 1
  data.loc[data['foundation_type'] != 'RC', 'F_RC'] = 0
  data.loc[data['foundation_type'] == 'Other', 'F_Other'] = 1
  data.loc[data['foundation_type'] != 'Other', 'F_Other'] = 0

data.drop(columns=['foundation_type'],  inplace=True)

data['ground_floor_type'].value_counts()
#This is categorical nominal data.

#Converting 'ground_floor_type' from 'object' to numerical values.
```

```
for i in data:
  data.loc[data['ground_floor_type'] == 'Mud', 'G_Mud'] = 1
  data.loc[data['ground_floor_type'] != 'Mud', 'G_Mud'] = 0
  data.loc[data['ground_floor_type'] == 'RC', 'G_RC'] = 1
  data.loc[data['ground_floor_type'] != 'RC', 'G_RC'] = 0
  data.loc[data['ground_floor_type'] == 'Brick/Stone', 'G_Brick/Stone'] = 1
  data.loc[data['ground_floor_type'] != 'Brick/Stone', 'G_Brick/Stone'] = 0
  data.loc[data['ground_floor_type'] == 'Timber', 'G_Timber'] = 1
  data.loc[data['ground_floor_type'] != 'Timber', 'G_Timber'] = 0
  data.loc[data['ground_floor_type'] == 'Other', 'G_Other'] = 1
  data.loc[data['ground_floor_type'] != 'Other', 'G_Other'] = 0

data.drop(columns=['ground_floor_type'],  inplace=True)

data['other_floor_type'].value_counts()

#Converting 'other_floor_type' from 'object' to numerical values.
for i in data:
  data.loc[data['other_floor_type'] == 'TImber/Bamboo-Mud', 'O_Timber'] = 1
  data.loc[data['other_floor_type'] != 'TImber/Bamboo-Mud', 'O_Timber'] = 0
  data.loc[data['other_floor_type'] == 'Not applicable', 'O_NA'] = 1
  data.loc[data['other_floor_type'] != 'Not applicable', 'O_NA'] = 0
  data.loc[data['other_floor_type'] == 'Timber-Planck', 'O_Planck'] = 1
  data.loc[data['other_floor_type'] != 'Timber-Planck', 'O_Planck'] = 0
  data.loc[data['other_floor_type'] == 'RCC/RB/RBC', 'O_RCC'] = 1
  data.loc[data['other_floor_type'] != 'RCC/RB/RBC', 'O_RCC'] = 0

data.drop(columns=['other_floor_type'],  inplace=True)

data['plan_configuration'].value_counts()

#Converting 'plan_configuration' from 'object' to numerical values.
for i in data:
  data.loc[data['plan_configuration'] == 'Rectangular', 'P_Rectangular'] = 1
  data.loc[data['plan_configuration'] != 'Rectangular', 'P_Rectangular'] = 0
  data.loc[data['plan_configuration'] == 'Square', 'P_Square'] = 1
  data.loc[data['plan_configuration'] != 'Square', 'P_Square'] = 0
  data.loc[data['plan_configuration'] == 'L-shape', 'P_L-shape'] = 1
  data.loc[data['plan_configuration'] != 'L-shape', 'P_L-shape'] = 0
  data.loc[data['plan_configuration'] == 'Multi-projected', 'P_Multi'] = 1
  data.loc[data['plan_configuration'] != 'Multi-projected', 'P_Multi'] = 0
  data.loc[data['plan_configuration'] == 'T-shape', 'P_T-shape'] = 1
  data.loc[data['plan_configuration'] != 'T-shape', 'P_T-shape'] = 0
  data.loc[data['plan_configuration'] == 'Others', 'P_Others'] = 1
  data.loc[data['plan_configuration'] != 'Others', 'P_Others'] = 0
  data.loc[data['plan_configuration'] == 'U-shape', 'P_U-shape'] = 1
  data.loc[data['plan_configuration'] != 'U-shape', 'P_U-shape'] = 0
  data.loc[data['plan_configuration'] == 'E-shape', 'P_E-shape'] = 1
  data.loc[data['plan_configuration'] != 'E-shape', 'P_E-shape'] = 0
```

```
  data.loc[data['plan_configuration'] == 'Building with Central Courtyard', 'P_BCC'] = 1
  data.loc[data['plan_configuration'] != 'Building with Central Courtyard', 'P_BCC'] = 0
  data.loc[data['plan_configuration'] == 'H-shape', 'P_H-shape'] = 1
  data.loc[data['plan_configuration'] != 'H-shape', 'P_H-shape'] = 0

data.drop(columns=['plan_configuration'],  inplace=True)

# Let's identify variables with a single value

no_variance_variables = []

for i in data.columns:
  if data[i].value_counts().values.shape[0] == 1:
    present_single_value = data[i].value_counts().index.values[0]
    print(f'Variable {i} has single value: {present_single_value}')
    no_variance_variables.append(i)

print(no_variance_variables)

data.describe()

data.shape

data.hist(column='age_building')

(data['age_building'] > 80).sum()

#As 'age_building' shows massive max values, all instances with a value more than a reasonable of less than 500 years, will be removed.
data.drop(data[data['age_building'] >= 80].index, inplace = True)

data.hist(column='age_building')

data.hist(column='plinth_area_sq_ft')

(data['plinth_area_sq_ft'] > 1500).sum()

data.drop(data[data['plinth_area_sq_ft'] >= 1500].index, inplace = True)

data.hist(column='plinth_area_sq_ft')

#Applying Max-Min scalling to the dataset, for a better performance in visualisation and experiments.
for i in data.columns:
    data[i] = (data[i] - data[i].min()) / (data[i].max() - data[i].min())

data.describe()

"""**Creating training, validation and test datasets**"""
```

#Splitting the dataset into train, validation (val) and test datasets.

```
train, test = train_test_split(data, test_size=0.4, random_state=20)

train.shape

test.shape

train,val = train_test_split(train,test_size=0.2, random_state=15)

train.shape

val.shape
```

#The result of this section are 3 datasets: training, validation and test, with 240352, 60089 and 200294 instances, respectively, and 33 features,
#where Building_ID is the unique identifier for every building and the feature TARGET is the label for each building showing two categorical values: collapse or safe.

"""EXPLORATORY ANALYSIS****"""**

```
train.info()
```

#Heatmap of the correlation matrix

```
plt.figure(figsize=(30,24))
plt.title('Correlation matrix for features in the trainig dataset', fontdict={'fontsize':8})
ax = sns.heatmap(train.corr(), annot=False, linewidths=0.1)
```

#List of correlated values between features and target.

```
train.corr()['Building_Collapse'].sort_values(ascending=False)
```

#Checking if the dataset is balanced

```
np.sum(train['Building_Collapse']) / train.shape[0]
```

#Distributon of the target values 'Building_Collapse'.

```
plt.figure(figsize=(10,5))
sns.countplot(data=train, x='Building_Collapse')
plt.title('Distribution of Building collapse values')
plt.xlabel('Building collapse: 1 if collapsing, 0 if stable')
plt.show()
```

#Histograms of the most 4 correlated features to target.

```
train.hist('has_superstructure_mud_mortar_stone')
train.hist('F_Mud')
train.hist('has_superstructure_cement_mortar_brick')
train.hist('G_RC')
```

#Histogram with target value by colour.

```
plt.figure(figsize=(7,4))
```

```
sns.countplot(x=train['has_superstructure_mud_mortar_stone'],hue=train['Building_Collapse'],palette='viridis')
plt.figure(figsize=(7,4))
sns.countplot(x=train['F_Mud'],hue=train['Building_Collapse'],palette='viridis')
plt.figure(figsize=(7,4))
sns.countplot(x=train['has_superstructure_cement_mortar_brick'],hue=train['Building_Collapse'],palette='viridis')
plt.figure(figsize=(7,4))
sns.countplot(x=train['G_RC'],hue=train['Building_Collapse'],palette='viridis')
```

#Percentage of values for the 4 most correlated variables with the target, that happen to be binary.

```
cols = [['has_superstructure_mud_mortar_stone', 'F_Mud'], ['has_superstructure_cement_mortar_brick',
      'G_RC']]

fig, axes = plt.subplots(ncols = 2, nrows = 2, figsize = (20,20))
for i, c in enumerate(cols):
   train[c[0]].value_counts().plot.pie(autopct='%.1f%%', ax = axes[i][0])
   train[c[1]].value_counts().plot.pie(autopct='%.1f%%', ax = axes[i][1])
plt.show()
```

"""FEATURE ENGINEERING.**"""**

 # Assigning datasets for calculation, splitting the training from target sets.

```
X_train = train.drop(columns=[ 'Building_Collapse'], axis=1)
y_train = train['Building_Collapse']
X_val = val.drop(columns=[ 'Building_Collapse'], axis=1)
y_val = val['Building_Collapse']
X_test = test.drop(columns=[ 'Building_Collapse'], axis=1)
y_test = test['Building_Collapse']

X_train.info()
```

#The numerical features are extracted and the remaining binary ones are converted into categorical data.

```
X_train_cat = X_train.drop(columns=['age_building','plinth_area_sq_ft','inclination', 'roof_strength',
'stability'],  inplace=False)
X_train_cat = X_train_cat.astype('category')

X_val_cat = X_val.drop(columns=['age_building','plinth_area_sq_ft','inclination', 'roof_strength',
'stability'],  inplace=False)
X_val_cat = X_val_cat.astype('category')

X_test_cat = X_test.drop(columns=['age_building','plinth_area_sq_ft','inclination', 'roof_strength',
'stability'],  inplace=False)
X_test_cat = X_test_cat.astype('category')

X_train_cat.info()
```

"""Multiple Correspondece Analysis - MCA"""

#The most correlative with target features are selected for the generation of the MCA.

```
X_train_cat_select=X_train_cat[{'has_superstructure_mud_mortar_stone', 'F_Mud', 'O_Timber', 'G_Mud',
'has_superstructure_bamboo','has_secondary_use_hotel', 'has_secondary_use_agriculture',
'has_superstructure_stone_flag', 'has_secondary_use_rental', 'G_RC', 'has_superstructure_timber',
'has_superstructure_cement_mortar_brick', 'F_Bamboo', 'O_NA', 'F_RC', 'O_RCC',
'has_superstructure_rc_engineered', 'F_Cement', 'has_superstructure_mud_mortar_brick',
'has_superstructure_rc_non_engineered'}]
X_val_cat_select=X_val_cat[{'has_superstructure_mud_mortar_stone', 'F_Mud', 'O_Timber', 'G_Mud',
'has_superstructure_bamboo','has_secondary_use_hotel', 'has_secondary_use_agriculture',
'has_superstructure_stone_flag', 'has_secondary_use_rental', 'G_RC', 'has_superstructure_timber',
'has_superstructure_cement_mortar_brick', 'F_Bamboo', 'O_NA', 'F_RC', 'O_RCC',
'has_superstructure_rc_engineered', 'F_Cement', 'has_superstructure_mud_mortar_brick',
'has_superstructure_rc_non_engineered'}]
X_test_cat_select=X_test_cat[{'has_superstructure_mud_mortar_stone', 'F_Mud', 'O_Timber', 'G_Mud',
'has_superstructure_bamboo','has_secondary_use_hotel', 'has_secondary_use_agriculture',
'has_superstructure_stone_flag', 'has_secondary_use_rental', 'G_RC', 'has_superstructure_timber',
'has_superstructure_cement_mortar_brick', 'F_Bamboo', 'O_NA', 'F_RC', 'O_RCC',
'has_superstructure_rc_engineered', 'F_Cement', 'has_superstructure_mud_mortar_brick',
'has_superstructure_rc_non_engineered'}]

X_train_cat_select.info()
```

#Generation of MCA with 2 dimensions for training, validation and test data.

```
train_mca = prince.MCA()
train_mca = train_mca.fit(X_train_cat_select)
train_mca = train_mca.transform(X_train_cat_select)

val_mca = prince.MCA()
val_mca = val_mca.fit(X_val_cat_select)
val_mca = val_mca.transform(X_val_cat_select)

test_mca = prince.MCA()
test_mca = test_mca.fit(X_test_cat_select)
test_mca = test_mca.transform(X_test_cat_select)

train_mca.info()

val_mca.info()

train_mca.describe()
```

#Applying Max-Min scalling to the mca datasets, to have the same scale of the rest of the features.

```
for i in train_mca.columns:
   train_mca[i] = (train_mca[i] - train_mca[i].min()) / (train_mca[i].max() - train_mca[i].min())

for i in val_mca.columns:
   val_mca[i] = (val_mca[i] - val_mca[i].min()) / (val_mca[i].max() - val_mca[i].min())
```

```
for i in test_mca.columns:
   test_mca[i] = (test_mca[i] - test_mca[i].min()) / (test_mca[i].max() - test_mca[i].min())

#Renaming the MCA features to string values, through copying the feautures.
train_mca['MCA-1']=train_mca[0]
train_mca['MCA-2']=train_mca[1]
train_mca.drop(columns=[0,1], inplace=True)
val_mca['MCA-1']=val_mca[0]
val_mca['MCA-2']=val_mca[1]
val_mca.drop(columns=[0,1], inplace=True)
test_mca['MCA-1']=test_mca[0]
test_mca['MCA-2']=test_mca[1]
test_mca.drop(columns=[0,1], inplace=True)

train_mca.describe()

#The original X_train columns and the MCA components are concatenated, creating a dataset with all features.
train_all = pd.concat([X_train, train_mca], axis="columns")
val_all = pd.concat([X_val, val_mca], axis="columns")
test_all = pd.concat([X_test, test_mca], axis="columns")

"""Autoencoder feature generation"""

tf.random.set_seed(30)
np.random.seed(30)

n = 5
_, input_neurons = X_train.shape

encoder = keras.models.Sequential([
   keras.layers.Dense(n, input_shape=[input_neurons]),
])
decoder = keras.models.Sequential([
   keras.layers.Dense(input_neurons, input_shape=[n]),
])

Autoencoder = keras.models.Sequential([encoder, decoder])
Autoencoder.compile(
   loss="mse",
   optimizer=keras.optimizers.Adam(learning_rate=0.001)
)

print(Autoencoder.summary())

history = Autoencoder.fit(
   X_train, X_train, epochs=10,
   validation_data=(X_val, X_val)
)
```

```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.savefig(
   'Autoencoder_train_loss.jpg',
   bbox_inches='tight',
)
plt.show()

ae_train_feature_array = np.array(encoder.predict(X_train))
ae_val_feature_array = np.array(encoder.predict(X_val))
ae_test_feature_array = np.array(encoder.predict(X_test))

ae_train = create_new_features_dataframe(ae_train_feature_array , "ae")
ae_val = create_new_features_dataframe(ae_val_feature_array , "ae")
ae_test = create_new_features_dataframe(ae_test_feature_array , "ae")

ae_train.describe()

"""Therefore, there are 4 types of datasets, depending on the features:
X_train/val/test containing the created original dataset.
train/val/_mca, with the mca components.
train/val/_all, that included all original features + MCA components.
train/val_ae, with the Autoencoder features.

Summary of general information of all datasets created.
"""

X_train.info()

train_mca.info()

train_all.info()

ae_train.info()

"""**Experiments.**"""

#Function to generate evaluation metrics report.
def evaluate_results(y, y_pred):
  """
  Evaluation Metrics:
  - Accuracy (Acc)
  - Recall
  - False Alarm (FAR): FP / N
```

```
 - Type II Error: FN / (FN + TP)
 - F1
 - MCC
 """
 tn, fp, fn, tp = confusion_matrix(y, y_pred).ravel()

 acc = accuracy_score(y, y_pred)
 recall = recall_score(y, y_pred)
 far = fp / (tn + fn)
 type_ii_err = fp / (fp + tn)
 f1 = f1_score(y, y_pred)
 mcc = matthews_corrcoef(y, y_pred)
 roc_auc = roc_auc_score(y, y_pred)

 print(f"Confussion Matrix:")
 print(f"\t{tp}\t{fn}")
 print(f"\t{fp}\t{tn}")

 print(f"Accuracy: {acc}")
 print(f"Detection Rate: {recall}")
 print(f"False Alarm Rate: {far}")
 print(f"Type II Error: {type_ii_err}")
 print(f"F1: {f1}")
 print(f"MCC: {mcc}")
 print(f"Roc auc: {roc_auc}")

 return dict(
    accuracy=acc,
    recall=recall,
    far=far,
    type_ii_err=type_ii_err,
    f1=f1,
    mcc=mcc,
    roc_auc=roc_auc,
 )

overall_results = [] #to reset the report list.

"""

Original features + Simple XGBoost Classifier
 """

hyperparams = dict(
 n_estimators=3,
 max_depth=3,
 random_state=30
)
```

```
performance = {}

performance["classifier"] = "Simple XGBoost"
performance["feature_gen"] = ""
performance["feature_sel"] = "Original features"
performance["hyperparams"] =str(hyperparams)

start_time = time.time()
model = xgb.XGBClassifier(**hyperparams)
time_to_process = time.time() - start_time

start_time = time.time()
model.fit(X_train, y_train)
time_to_fit = time.time() - start_time

print('\nTrain Dataset\n')
performance['train'] = evaluate_results(y_train, model.predict(X_train));
print('\nVal Dataset\n')
performance['val'] = evaluate_results(y_val, model.predict(X_val));

summary = pd.DataFrame(performance)
overall_results.append(summary)

print('\nSUMMARY')
print(summary)
print(time_to_fit)
print(time_to_process)

sns.set(font_scale=1)
img = summary.plot.bar(subplots=False, grid=True)
img.figure.savefig(

f'{performance["classifier"]}_{performance["feature_gen"]}_{performance["feature_sel"]}_overfited.jpg',
   bbox_inches='tight',
)

"""
 Original features + Linear booster XGBoost Classifier"""

hyperparams = dict(
  n_estimators=3,
  max_depth=3,
  booster='gblinear',
  random_state=30
)

performance = {}

performance["classifier"] = "Linear XGBoost"
```

```
performance["feature_gen"] = ""
performance["feature_sel"] = "Original features"
performance["hyperparams"] =str(hyperparams)

start_time = time.time()
model = xgb.XGBClassifier(**hyperparams)
time_to_process = time.time() - start_time

start_time = time.time()
model.fit(X_train, y_train)
time_to_fit = time.time() - start_time


print('\nTrain Dataset\n')
performance['train'] = evaluate_results(y_train, model.predict(X_train));
print('\nVal Dataset\n')
performance['val'] = evaluate_results(y_val, model.predict(X_val));

summary = pd.DataFrame(performance)
overall_results.append(summary)

print('\nSUMMARY')
print(summary)
print(time_to_fit)
print(time_to_process)

sns.set(font_scale=1)
img = summary.plot.bar(subplots=False, grid=True)
img.figure.savefig(
   f'{performance["classifier"]}_{performance["feature_gen"]}_{performance["feature_sel"]}.jpg',
   bbox_inches='tight',
)
```

""" MCA components + Simple XGBoost Classifier (as it seems to perform better than linear XGBoost)"""

```
hyperparams = dict(
  n_estimators=3,
  max_depth=3,
  random_state=30
)

performance = {}

performance["classifier"] = "XGBoost"
performance["feature_gen"] = "MCA"
performance["feature_sel"] = "MCA"
performance["hyperparams"] =str(hyperparams)

start_time = time.time()
```

```
model = xgb.XGBClassifier(**hyperparams)
time_to_process = time.time() - start_time


start_time = time.time()
model.fit(train_mca, y_train)
time_to_fit = time.time() - start_time

print('\nTrain Dataset\n')
performance['train'] = evaluate_results(y_train, model.predict(train_mca));
print('\nVal Dataset\n')
performance['val'] = evaluate_results(y_val, model.predict(val_mca));

summary = pd.DataFrame(performance)
overall_results.append(summary)

print('\nSUMMARY')
print(summary)
print(time_to_fit)
print(time_to_process)

sns.set(font_scale=1)
img = summary.plot.bar(subplots=False, grid=True)
img.figure.savefig(
   f'{performance["classifier"]}_{performance["feature_gen"]}_{performance["feature_sel"]}.jpg',
   bbox_inches='tight',
)

""" MCA + original features + Simple XGBoost Classifier"""

hyperparams = dict(
  n_estimators=3,
  max_depth=3,
  random_state=30
)

performance = {}

performance["classifier"] = "XGBoost"
performance["feature_gen"] = "MCA"
performance["feature_sel"] = "MCA + Original features"
performance["hyperparams"] =str(hyperparams)


start_time = time.time()
model = xgb.XGBClassifier(**hyperparams)
time_to_process = time.time() - start_time

start_time = time.time()
```

```
model.fit(train_all, y_train)
time_to_fit = time.time() - start_time

print('\nTrain Dataset\n')
performance['train'] = evaluate_results(y_train, model.predict(train_all));
print('\nVal Dataset\n')
performance['val'] = evaluate_results(y_val, model.predict(val_all));

summary = pd.DataFrame(performance)
overall_results.append(summary)

print('\nSUMMARY')
print(summary)
print(time_to_fit)
print(time_to_process)

sns.set(font_scale=1)
img = summary.plot.bar(subplots=False, grid=True)
img.figure.savefig(
   f'{performance["classifier"]}_{performance["feature_gen"]}_{performance["feature_sel"]}.jpg',
   bbox_inches='tight',
)
```

"""MCA components + SVM classfier"""

```
start_time = time.time()
model = LinearSVC()
time_to_process = time.time() - start_time

start_time = time.time()
model.fit(train_mca, y_train)
time_to_fit = time.time() - start_time

performance = {}

hyperparams = model.get_params()

performance["classifier"] = "SVM"
performance["feature_gen"] = "MCA"
performance["feature_sel"] = "MCA"
performance["hyperparams"] =str(hyperparams)

print('\nTrain Dataset\n')
performance['train'] = evaluate_results(y_train, model.predict(train_mca));
print('\nVal Dataset\n')
performance['val'] = evaluate_results(y_val, model.predict(val_mca));

summary = pd.DataFrame(performance)
overall_results.append(summary)
```

```
print('\nSUMMARY')
print(summary)
print(time_to_fit)
print(time_to_process)

sns.set(font_scale=1)
img = summary.plot.bar(subplots=False, grid=True)
img.figure.savefig(
   f'{performance["classifier"]}_{performance["feature_gen"]}_{performance["feature_sel"]}.jpg',
   bbox_inches='tight',
)

"""Autoencoder features + Simple XGBoost Classifier"""

hyperparams = dict(
  n_estimators=3,
  max_depth=3,
  random_state=30
)

performance = {}

performance["classifier"] = "XGBoost"
performance["feature_gen"] = "Autoencoder"
performance["feature_sel"] = "Autoencoder"
performance["hyperparams"] =str(hyperparams)


start_time = time.time()
model = xgb.XGBClassifier(**hyperparams)
time_to_process = time.time() - start_time

start_time = time.time()
model.fit(ae_train, y_train)
time_to_fit = time.time() - start_time

print('\nTrain Dataset\n')
performance['train'] = evaluate_results(y_train, model.predict(ae_train));
print('\nVal Dataset\n')
performance['val'] = evaluate_results(y_val, model.predict(ae_val));

summary = pd.DataFrame(performance)
overall_results.append(summary)

print('\nSUMMARY')
print(summary)
print(time_to_fit)
print(time_to_process)
```

```
sns.set(font_scale=1)
img = summary.plot.bar(subplots=False, grid=True)
img.figure.savefig(

f'{performance["classifier"]}_{performance["feature_gen"]}_{performance["feature_sel"]}_overfited.jpg',
    bbox_inches='tight',
)

"""Autoencoder + SVM classifier"""

start_time = time.time()
model = LinearSVC()
time_to_process = time.time() - start_time

start_time = time.time()
model.fit(ae_train, y_train)
time_to_fit = time.time() - start_time

performance = {}

hyperparams = model.get_params()

performance["classifier"] = "SVM"
performance["feature_gen"] = "Autoencoder"
performance["feature_sel"] = "Autoencoder"
performance["hyperparams"] =str(hyperparams)

print('\nTrain Dataset\n')
performance['train'] = evaluate_results(y_train, model.predict(ae_train));
print('\nVal Dataset\n')
performance['val'] = evaluate_results(y_val, model.predict(ae_val));

summary = pd.DataFrame(performance)
overall_results.append(summary)

print('\nSUMMARY')
print(summary)
print(time_to_fit)
print(time_to_process)

sns.set(font_scale=1)
img = summary.plot.bar(subplots=False, grid=True)
img.figure.savefig(
    f'{performance["classifier"]}_{performance["feature_gen"]}_{performance["feature_sel"]}.jpg',
    bbox_inches='tight',
)

"""Original features + ANN (multilayer perceptron - MLP) classifier"""
```

```
hyperparams = dict(
    hidden_layer_sizes=(
       5,
    ),
    random_state=1
)

performance = {}

performance["classifier"] = "ANN"
performance["feature_gen"] = ""
performance["feature_sel"] = "Original features"
performance["hyperparams"] =str(hyperparams)


start_time = time.time()
model = MLPClassifier(**hyperparams)
time_to_process = time.time() - start_time

start_time = time.time()
model.fit(X_train, y_train)
time_to_fit = time.time() - start_time

print('\nTrain Dataset\n')
performance['train'] = evaluate_results(y_train, model.predict(X_train));
print('\nVal Dataset\n')
performance['val'] = evaluate_results(y_val, model.predict(X_val));

summary = pd.DataFrame(performance)
overall_results.append(summary)

print('\nSUMMARY')
print(summary)
print(time_to_fit)
print(time_to_process)

sns.set(font_scale=1)
img = summary.plot.bar(subplots=False, grid=True)
img.figure.savefig(
   f'{performance["classifier"]}_{performance["feature_gen"]}_{performance["feature_sel"]}.jpg',
   bbox_inches='tight',
)
```

"""Autoencoder features + ANN (multilayer perceptron - MLP) classifier"""

```
hyperparams = dict(
    hidden_layer_sizes=(
       5,
```

```
    ),
    random_state=1
)

performance = {}

performance["classifier"] = "ANN"
performance["feature_gen"] = "Autoencoder"
performance["feature_sel"] = "Autoencoder"
performance["hyperparams"] =str(hyperparams)

start_time = time.time()
model = MLPClassifier(**hyperparams)
time_to_process = time.time() - start_time

start_time = time.time()
model.fit(ae_train, y_train)
time_to_fit = time.time() - start_time

print('\nTrain Dataset\n')
performance['train'] = evaluate_results(y_train, model.predict(ae_train));
print('\nVal Dataset\n')
performance['val'] = evaluate_results(y_val, model.predict(ae_val));

summary = pd.DataFrame(performance)
overall_results.append(summary)

print('\nSUMMARY')
print(summary)
print(time_to_fit)
print(time_to_process)

sns.set(font_scale=1)
img = summary.plot.bar(subplots=False, grid=True)
img.figure.savefig(
   f'{performance["classifier"]}_{performance["feature_gen"]}_{performance["feature_sel"]}.jpg',
   bbox_inches='tight',
)

"""Original features with Decision tree

"""

hyperparams = dict(max_features = None,
                   max_depth = 45,
                   min_samples_split = 3,
                   min_samples_leaf = 30,
                   random_state=30
)
```

```
performance = {}

performance["classifier"] = "Decision tree"
performance["feature_gen"] = ""
performance["feature_sel"] = "Original features"
performance["hyperparams"] =str(hyperparams)

start_time = time.time()
model = DecisionTreeClassifier(**hyperparams)
time_to_process = time.time() - start_time

start_time = time.time()
model.fit(X_train, y_train)
time_to_fit = time.time() - start_time

print('\nTrain Dataset\n')
performance['train'] = evaluate_results(y_train, model.predict(X_train));
print('\nVal Dataset\n')
performance['val'] = evaluate_results(y_val, model.predict(X_val));

summary = pd.DataFrame(performance)
overall_results.append(summary)

print('\nSUMMARY')
print(summary)
print(time_to_fit)
print(time_to_process)

sns.set(font_scale=1)
img = summary.plot.bar(subplots=False, grid=True)
img.figure.savefig(
   f'{performance["classifier"]}_{performance["feature_gen"]}_{performance["feature_sel"]}.jpg',
   bbox_inches='tight',
)

"""MCA with Decision tree

"""

hyperparams = dict(max_features = None,
                   max_depth = 45,
                   min_samples_split = 3,
                   min_samples_leaf = 30,
                   random_state=30
)

performance = {}
```

```
performance["classifier"] = "Decision tree"
performance["feature_gen"] = "MCA"
performance["feature_sel"] = "MCA"
performance["hyperparams"] =str(hyperparams)

start_time = time.time()
model = DecisionTreeClassifier(**hyperparams)
time_to_process = time.time() - start_time

start_time = time.time()
model.fit(train_mca, y_train)
time_to_fit = time.time() - start_time

print('\nTrain Dataset\n')
performance['train'] = evaluate_results(y_train, model.predict(train_mca));
print('\nVal Dataset\n')
performance['val'] = evaluate_results(y_val, model.predict(val_mca));

summary = pd.DataFrame(performance)
overall_results.append(summary)

print('\nSUMMARY')
print(summary)
print(time_to_fit)
print(time_to_process)

sns.set(font_scale=1)
img = summary.plot.bar(subplots=False, grid=True)
img.figure.savefig(
   f'{performance["classifier"]}_{performance["feature_gen"]}_{performance["feature_sel"]}.jpg',
   bbox_inches='tight',
)
```

"""MCA + original features with Decision tree"""

```
hyperparams = dict(max_features = None,
                   max_depth = 45,
                   min_samples_split = 3,
                   min_samples_leaf = 30,
                   random_state=30
)

performance = {}

performance["classifier"] = "Decision tree"
performance["feature_gen"] = "MCA"
performance["feature_sel"] = "MCA + Original features"
performance["hyperparams"] =str(hyperparams)
```

```
start_time = time.time()
model = DecisionTreeClassifier(**hyperparams)
time_to_process = time.time() - start_time

start_time = time.time()
model.fit(train_all, y_train)
time_to_fit = time.time() - start_time

print('\nTrain Dataset\n')
performance['train'] = evaluate_results(y_train, model.predict(train_all));
print('\nVal Dataset\n')
performance['val'] = evaluate_results(y_val, model.predict(val_all));

summary = pd.DataFrame(performance)
overall_results.append(summary)

print('\nSUMMARY')
print(summary)
print(time_to_fit)
print(time_to_process)

sns.set(font_scale=1)
img = summary.plot.bar(subplots=False, grid=True)
img.figure.savefig(
   f'{performance["classifier"]}_{performance["feature_gen"]}_{performance["feature_sel"]}.jpg',
   bbox_inches='tight',
)

"""Autoencoder features with Decision tree

"""

hyperparams = dict(max_features = None,
                   max_depth = 45,
                   min_samples_split = 3,
                   min_samples_leaf = 30,
                   random_state=30
)

performance = {}

performance["classifier"] = "Decision tree"
performance["feature_gen"] = "Autoencoder"
performance["feature_sel"] = "Autoencoder"
performance["hyperparams"] =str(hyperparams)

start_time = time.time()
model = DecisionTreeClassifier(**hyperparams)
time_to_process = time.time() - start_time
```

```
start_time = time.time()
model.fit(ae_train, y_train)
time_to_fit = time.time() - start_time

print('\nTrain Dataset\n')
performance['train'] = evaluate_results(y_train, model.predict(ae_train));
print('\nVal Dataset\n')
performance['val'] = evaluate_results(y_val, model.predict(ae_val));

summary = pd.DataFrame(performance)
overall_results.append(summary)

print('\nSUMMARY')
print(summary)
print(time_to_fit)
print(time_to_process)

sns.set(font_scale=1)
img = summary.plot.bar(subplots=False, grid=True)
img.figure.savefig(
   f'{performance["classifier"]}_{performance["feature_gen"]}_{performance["feature_sel"]}.jpg',
   bbox_inches='tight',
)
```

"""Original features with Random Forest"""

```
hyperparams = dict(max_features = None,
                   max_depth = 45,
                   min_samples_split = 3,
                   min_samples_leaf = 30,
                   random_state=30)

performance = {}

performance["classifier"] = "Random Forest"
performance["feature_gen"] = ""
performance["feature_sel"] = "Original features"
performance["hyperparams"] =str(hyperparams)

start_time = time.time()
model = RandomForestClassifier(**hyperparams)
time_to_process = time.time() - start_time

start_time = time.time()
model.fit(X_train, y_train)
time_to_fit = time.time() - start_time

print('\nTrain Dataset\n')
```

```
performance['train'] = evaluate_results(y_train, model.predict(X_train));
print('\nVal Dataset\n')
performance['val'] = evaluate_results(y_val, model.predict(X_val));

summary = pd.DataFrame(performance)
overall_results.append(summary)

print('\nSUMMARY')
print(summary)
print(time_to_fit)
print(time_to_process)

sns.set(font_scale=1)
img = summary.plot.bar(subplots=False, grid=True)
img.figure.savefig(
   f'{performance["classifier"]}_{performance["feature_gen"]}_{performance["feature_sel"]}.jpg',
   bbox_inches='tight',
)
```

"""Original features + MCA with Random Forest"""

```
hyperparams = dict(max_features = None,
                   max_depth = 45,
                   min_samples_split = 3,
                   min_samples_leaf = 30,
                   random_state=30)

performance = {}

performance["classifier"] = "Random Forest"
performance["feature_gen"] = "MCA"
performance["feature_sel"] = "MCA + Original features"
performance["hyperparams"] =str(hyperparams)

start_time = time.time()
model = RandomForestClassifier(**hyperparams)
time_to_process = time.time() - start_time

start_time = time.time()
model.fit(train_all, y_train)
time_to_fit = time.time() - start_time

print('\nTrain Dataset\n')
performance['train'] = evaluate_results(y_train, model.predict(train_all));
print('\nVal Dataset\n')
performance['val'] = evaluate_results(y_val, model.predict(val_all));

summary = pd.DataFrame(performance)
overall_results.append(summary)
```

```
print('\nSUMMARY')
print(summary)
print(time_to_fit)
print(time_to_process)

sns.set(font_scale=1)
img = summary.plot.bar(subplots=False, grid=True)
img.figure.savefig(
   f'{performance["classifier"]}_{performance["feature_gen"]}_{performance["feature_sel"]}.jpg',
   bbox_inches='tight',
)

pd.concat(overall_results).loc["f1"].sort_values(by="val", ascending=False)

pd.concat(overall_results).loc["recall"].sort_values(by="val", ascending=False)

pd.concat(overall_results).loc["accuracy"].sort_values(by="val", ascending=False)

pd.concat(overall_results).loc["type_ii_err"].sort_values(by="val", ascending=False)

"""**Selected algorthims for testing**

Original features + Linear booster XGBoost Classifier
"""

hyperparams = dict(
  n_estimators=3,
  max_depth=3,
  booster='gblinear',
  random_state=30
)

performance = {}

performance["classifier"] = "Linear XGBoost"
performance["feature_gen"] = ""
performance["feature_sel"] = "Original features"
performance["hyperparams"] =str(hyperparams)

start_time = time.time()
model = xgb.XGBClassifier(**hyperparams)
time_to_process = time.time() - start_time

start_time = time.time()
model.fit(X_train, y_train)
time_to_fit = time.time() - start_time
```

```
print('\nTrain Dataset\n')
performance['train'] = evaluate_results(y_train, model.predict(X_train));
print('\nTest Dataset\n')
performance['test'] = evaluate_results(y_test, model.predict(X_test));

summary = pd.DataFrame(performance)
overall_results.append(summary)

print('\nSUMMARY')
print(summary)
print(time_to_fit)
print(time_to_process)

sns.set(font_scale=1)
img = summary.plot.bar(subplots=False, grid=True)
img.figure.savefig(
   f'{performance["classifier"]}_{performance["feature_gen"]}_{performance["feature_sel"]}.jpg',
   bbox_inches='tight',
)

"""Autoencoder + SVM classifier"""

start_time = time.time()
model = LinearSVC()
time_to_process = time.time() - start_time

start_time = time.time()
model.fit(ae_train, y_train)
time_to_fit = time.time() - start_time

performance = {}

hyperparams = model.get_params()

performance["classifier"] = "SVM"
performance["feature_gen"] = "Autoencoder"
performance["feature_sel"] = "Autoencoder"
performance["hyperparams"] =str(hyperparams)

print('\nTrain Dataset\n')
performance['train'] = evaluate_results(y_train, model.predict(ae_train));
print('\nTest Dataset\n')
performance['test'] = evaluate_results(y_test, model.predict(ae_test));

summary = pd.DataFrame(performance)
overall_results.append(summary)

print('\nSUMMARY')
print(summary)
```

```
print(time_to_fit)
print(time_to_process)

sns.set(font_scale=1)
img = summary.plot.bar(subplots=False, grid=True)
img.figure.savefig(
   f'{performance["classifier"]}_{performance["feature_gen"]}_{performance["feature_sel"]}.jpg',
   bbox_inches='tight',
)

"""MCA with Decision tree

"""

hyperparams = dict(max_features = None,
                   max_depth = 45,
                   min_samples_split = 3,
                   min_samples_leaf = 30,
                   random_state=30
)

performance = {}

performance["classifier"] = "Decision tree"
performance["feature_gen"] = "MCA"
performance["feature_sel"] = "MCA"
performance["hyperparams"] =str(hyperparams)

start_time = time.time()
model = DecisionTreeClassifier(**hyperparams)
time_to_process = time.time() - start_time

start_time = time.time()
model.fit(train_mca, y_train)
time_to_fit = time.time() - start_time

print('\nTrain Dataset\n')
performance['train'] = evaluate_results(y_train, model.predict(train_mca));
print('\nTest Dataset\n')
performance['test'] = evaluate_results(y_test, model.predict(test_mca));

summary = pd.DataFrame(performance)
overall_results.append(summary)

print('\nSUMMARY')
print(summary)
print(time_to_fit)
print(time_to_process)
```

```
sns.set(font_scale=1)
img = summary.plot.bar(subplots=False, grid=True)
img.figure.savefig(
   f'{performance["classifier"]}_{performance["feature_gen"]}_{performance["feature_sel"]}.jpg',
   bbox_inches='tight',
)

"""Original features with Random Forest"""

hyperparams = dict(max_features = None,
                   max_depth = 45,
                   min_samples_split = 3,
                   min_samples_leaf = 30,
                   random_state=30)

performance = {}

performance["classifier"] = "Random Forest"
performance["feature_gen"] = ""
performance["feature_sel"] = "Original features"
performance["hyperparams"] =str(hyperparams)

start_time = time.time()
model = RandomForestClassifier(**hyperparams)
time_to_process = time.time() - start_time

start_time = time.time()
model.fit(X_train, y_train)
time_to_fit = time.time() - start_time

print('\nTrain Dataset\n')
performance['train'] = evaluate_results(y_train, model.predict(X_train));
print('\nTest Dataset\n')
performance['test'] = evaluate_results(y_test, model.predict(X_test));

summary = pd.DataFrame(performance)
overall_results.append(summary)

print('\nSUMMARY')
print(summary)
print(time_to_fit)
print(time_to_process)

sns.set(font_scale=1)
img = summary.plot.bar(subplots=False, grid=True)
img.figure.savefig(
   f'{performance["classifier"]}_{performance["feature_gen"]}_{performance["feature_sel"]}.jpg',
   bbox_inches='tight',
)
```