# Modeling Earthquake Damage

**Beyza Cevik** [1] **Mert Cokelek** [1] **Mustafa Sercan Amac** [1]

## Abstract

Earthquakes around the world have killed millions of people and more millions became homeless. The impacts of this deadliest natural disaster include structural damage to buildings, fires, initiation of slope failures, liquefaction and tsunami. Earthquakes are very difficult to predict. Therefore, in this study, we propose methods to predict the damage level to a building to decrease the death toll and economic loss. A variety of machine learning techniques are considered for the classification algorithm such as K-Nearest Neighbors, Neural Networks and Random Forest etc. We do experiment on a dataset with 260,601 buildings and 38 features. Our goal is to make damage-level predictions on buildings, based on their specifications. We achieved a f1 score of 0.7456 with XGBoost on Richter's Predictor: Modeling Earthquake Damage challenge where the highest score is 0.7558.

## 1. Introduction

Humanity has faced a lot of devastating earthquakes since ancient times. Earthquakes tend to have fatal consequences as people began to live in buildings. Engineers design more resistant or even earthquake-proof buildings to reduce the severe consequences of an unforeseen earthquake. However, neither the newest-constructed buildings are durable to such powerful disasters nor every building has these new engineering qualifications. We can not prevent an earthquake, but we can have sight about its possible damage level on the buildings, and make people take precautions for possible disasters. Hence, repairing and stabilizing the building can primarily keep people from being killed by an earthquake and secondarily can minimize financial loss as a result of knowing about which level the building can be damaged by a possible earthquake. The algorithms we used predict the post-earthquake damage level on a building. There are three damage level states namely, low damage level, medium damage level, and high damage level. Classifying the damage level of a given structure has a low fault-tolerance, thus it requires a very accurate model. For this purpose, at first, we approached with fundamental machine learning methods: K-Nearest Neighbors (K-NN), Multinomial Logistic Regression, Naive Bayes, Random Forest Algorithm, Neural Networks and boosting methods.

## 2. Related Work

Investigation on damage level predictions of earthquakes has been an active research field because it can greatly reduce the extensive loss of life and property. There are many studies(such as [11],[13],[14]) of civil engineers and geologists. However, machine learning techniques are a rather recent application. There are several attempts ([5], [4], [9]) to predict earthquake damage levels using machine learning and deep learning methods. Dissimilar to our study there is a study[10] for predicting damage level utilizing post-earthquake photos. However, there are also a few recent studies closer to ours. Decision tree learning algorithms used in one of these studies[5]. This study and our model both require structural properties as input but unlike our model, it requires a few earthquake characteristics. They use two decision trees for damage prediction for regular reinforced concrete buildings. The first decision tree decides whether damage occurs in an RC building. Also, the second decision tree determines the severity of the damage state. The main focus of one of the most recent studies[4] is the importance of features affecting earthquake fatalities. This study implements a deep learning model based on selected features for predicting seismic fatalities. They used Random Forest, classification and regression tree (CART) model, and AdaBoost model to evaluate the importance of features and they observed that the Random Forest model was better than the other models. This work indicated that the deep learning model in this study performed well for predicting seismic fatalities. Our study is related to this study as we implemented basic algorithms for comparison.

## 3. The Approach

### 3.1. Dataset

The data we will use is hosted at [1] and is from the earthquake happened in Gorkha, Nepal. It consists of 260,601 buildings and 38 features such as age, height, area, land surface percentage, number of floors, etc. which are labeled as 1,2,3; corresponding to low-medium-high levels of damage,

respectively. There is class imbalance as it is visualized in Figure 1

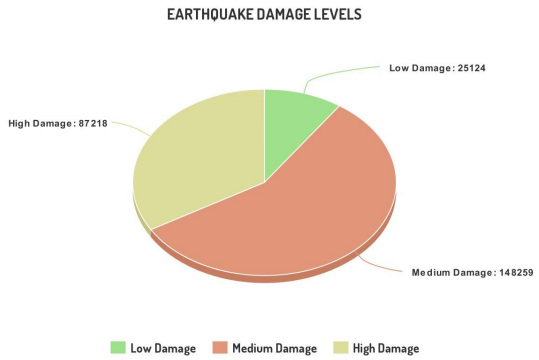


*Figure 1.* Distrubution of labels

Predicting damage may occur on a building just by knowing its features is quite hard since this is a very complex problem that requires lots of parameters than just the building features. Since there is no known approach to solve this problem, we will first apply basic algorithms such as K-NN, Naive Bayes, Logistic Regression and the Random Forest. We investigate if this problem is solvable by machine learning algorithms by examining how these algorithms behave. In our investigations, we observed a huge class imbalance in the data which can be seen in Figure 1. We will discuss the results of these algorithms in experiments section. Our main approach is to train a deep neural network model on this problem. However, we first need to think about class imbalance problems to prevent overfitting. There are many approaches to solve the class imbalance problem while training a neural network such as focal loss and weighted cross entropy loss. These approaches are explained in the sections below.

### 3.2. Stochastic Gradient Descent with momentum

Stochastic Gradient Descent (SGD)[8] is one of the most widely used optimization algorithms for training neural networks. However, it has an oscillation problem while reaching the optima that prevents the model from reaching the optima. A solution proposed for this problem is using a momentum variable [12] to reduce the oscillation.

$$\begin{aligned} v_t &= \gamma v_{t-1} + \eta \nabla_\theta J(\theta) \\ \theta &= \theta - v_t \end{aligned} \tag{1}$$

We set the momentum $\gamma$ = 0.9 in our experiments. The momentum increases for the gradients are in the same direction, and decreases for the gradients that change the direction. So we get fewer oscillations and faster convergence.

### 3.3. Adaptive Moment Estimation

Adaptive Moment Estimation (ADAM) [6] was proposed in 2014 and became one of the most popular optimization algorithms for training deep neural networks. It has a faster convergence than SGD with Momentum. It keeps both exponentially decaying average of past squared gradients $v\_t$ and exponentially decaying average of past gradients $m\_t$ with the equations below:

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1 - \beta_1)\, g_t \\ v_t &= \beta_2 v_{t-1} + (1 - \beta_2)\, g_t^2 \end{aligned} \tag{2}$$

With these variables the update rule for ADAM is:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t \tag{3}$$

We used value of 0.0001 for weight decay , 0.9 for $\beta$_1, 0.999 for $\beta$_2 and 1e-08 for $\epsilon$ in our experiments.

### 3.4. Step Learning Rate

Step Learning rate (StepLR) is a very simple method for decreasing the learning rate. It has 2 parameters factor and steps size. The learning rate is decreased by a factor of $\alpha$ after every step size of epochs. We used 0.1 for $\alpha$ and 24 for step size in our experiments.

### 3.5. Choosing a Good Learning Rate

Choosing a good learning rate is a very hard and time consuming problem. To overcome this problem we used a learning rate finding technique from fast.ai [3] library. It consists of parameters start lr, end lr and number of iterations. From the learning rate-loss graph we choose the learning rate from the middle of the sharpest slope.

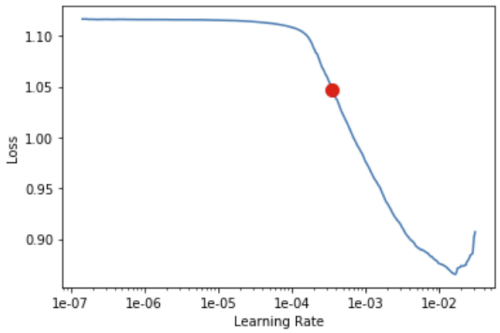


*Figure 2.* An example plot for choosing the learning rate.

### 3.6. Weighted Cross Entropy Loss

Cross-Entropy Loss [2] is one of the most popular loss functions for training neural networks. But if you have

a class imbalance problem you can use Weighted Cross-Entropy Loss (WCE) to obtain a better result. We basically create a vector of weights **w** for each class and we define the WCE as:

$$\mathcal{L}_n(\mathbf{W}) = -w_{c_n} \log \mathbf{y}_{c_n}(\mathbf{x}_n, \mathbf{W}) \tag{4}$$

We have 3 classes and we used weights of 1.0,0.7,0.8 for our experiments.

### 3.7. Focal Loss

Focal Loss[7] is a loss function to reduce the problem of class imbalance with simple modifications to the Cross-Entropy Loss. Focal loss is defined as:

$$FL(p_t) = -(1-p_t)^{\gamma} \log(p_t) \tag{5}$$

Here if p is defined as the probability distribution on the classes calculated by the model :

$$p_t = \begin{cases} p & \text{if } y = 1 \\ 1-p & \text{else} \end{cases} \tag{6}$$

In equation 5 the term $(1-p_t)^{\gamma}$ is added to the standard cross-entropy loss. By adding this term focal loss is decreasing the loss coming from the labels that are easily learned and makes the model focus on hard to learn labels.

### 3.8. Stochastic Gradient Descent with Warm Restarts

While training Deep Learning models, it is very common for the model to stuck in local optima. In this context, one of the suggested approaches to reach better optima in the parameters space is the use of restarting Stochastic Gradient Descent strategies. In Stochastic Gradient Descent with Warm Restarts -which is one of these strategies- suggested in [8], learning is being restarted with some values, quickly and periodically and decremented on schedule. These restart and value assignments for $i$'th run are done according to the equation below:

$$\eta_t = \eta^i_{min} + \frac{1}{2}(\eta^i_{max} - \eta^i_{min})\left(1 + \cos\left(\frac{T_{cur}}{T_i}\pi\right)\right) \tag{7}$$

Here, $T_{cur}$ corresponds to the number of epochs after the last restart, where $T_i$ corresponds to the number of epochs of the current restart and at every restart, $T_{mul}$ is incremented by the coefficent, represented with:

$$T_i = T_{cur} + T_i * T_{mul} \tag{8}$$

### 3.9. Reduce Learning Rate on Plateau

While training neural networks, one of the biggest problems is when to decay the learning rate. An intuitive approach for that problem is to reduce the learning rate by a factor of $\alpha$ if the metric we use on the validation set did not get better for n times. Here we call n as patience and we update the learning rate if the patience is exceeded. We used 0.1 for $\alpha$ and 5,10 for patience in our experiments.

## 4. Experimental Results

Dataset Description: The dataset contains damage levels of 260,000+ buildings with 38 features. Most of the features have discrete values, however, there are 8 features that are categorical. First, we converted these features into discrete forms, e.g 3 distinct categories of a feature took values "0, 1, 2". After conversion, the dataset is split into Train (80%) and Test (20%) sets.

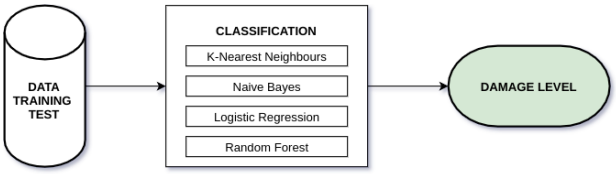


*Figure 3.* Flowchart of our experiments

### 4.1. K-nearest Neighbours Algorithm

After preparing the dataset, the K-NN classifier is applied and it resulted in f1 score of 0.7030. Then, weighted K-NN resulted in 0.7073. For weighted K-NN, euclidian distance is used as the distance metric.

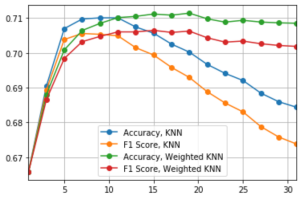


*Figure 4.* Accuracy,F1-Depth Graph for different K values

We then applied resampling using SMOTETomek technique. But it didnt result in a better score. On the contrary it made it worse. As it can be seen from the figure below.

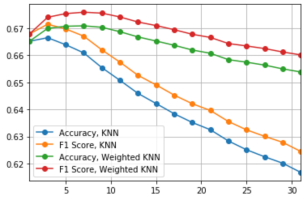


*Figure 5.* Accuracy,F1-Depth Graph for different K values

The reason for using K-NN is it is very simple to apply, especially in the beginning, to see if there is a pattern between features and labels; also it has few parameters to tune (number of neighbors, distance metric); and it doesn't require the classes to be linearly separable.

### 4.2. Naive Bayes

We applied Gaussian Naive Bayes Classifier as our second approach. Each sample was taken as a distinct vector of length 38, and they were assumed conditionally independent and coming from a normal distribution. This approach was not successful and resulted in 46.7% accuracy and 0.3410 f1 score. There are 2 major reasons for this outcome, one of these is some of the features are boolean and not coming from a normal distribution,

### 4.3. Logistic Regression

Our third approach was using Multinomial Logistic Regression (LR). The reason we choose LR is that it is highly interpretable, it doesn't require features to be scaled, doesn't need any tuning and its outputs are well-discriminated probabilities. However, the LR could not reach K-NN's performance (57.3% accuracy and 0.4923 f1 score). The main reason behind this is "class imbalance". When we look at the predictions of the model we observed that most of the predictions are made on medium level damage. This shows that class imbalance is very effective in the predictions of the logistic regression model.

### 4.4. Random Forest

Random Forest is another machine learning method that operates by constructing multiple decision trees. The final decision is made based on the majority of the trees and is chosen by the random forest. The advantages of using a random forest algorithm are; it reduces the overfitting, gives higher accuracy than 1 decision tree since it reduces the variance by ensembling different decision trees, and runs efficiently on large datasets. In our experiments, the random forest resulted in f1 score of 0.7187 with max depth = 30. We observed that the higher depth results in higher training performance, for the random forest model. However, it tends to overfit in out dataset since it fits the samples in the training data to leaves most correctly and in a noisy dataset. Outliers can effect the structure of the trees. Our goal is to find the best f1 score on our test set. For this purpose, initially we investigated the effects of "max. depth of trees" on performance. Here is the corresponding "max. depth – f1 score" plot 6. In the plot 6, we can see that after the values of max. depth 30, the model starts to overfit. So, in our next experiments we will tune the parameters based on this value. In the table 1, the relationships between "n_estimators – f1 scores" are shown. Number of estimators

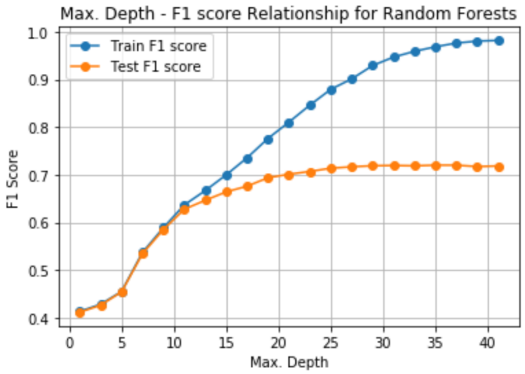


*Figure 6.* Training of a random forest model by adjusting max depth orange curve is for training set, blue curve is for test set.

corresponds to the total number of weak learners (stamps) used in decision. From the table it can be seen that the more

| max_depth | n_estimators | f1_train | f1_test |
|---|---|---|---|
| 30 | 25 | 0.9300 | 0.7118 |
| 30 | 50 | 0.9335 | 0.7175 |
| 30 | 100 | 0.9341 | 0.7187 |
| 30 | 250 | 0.9388 | 0.7203 |
| 30 | 500 | 0.9382 | 0.7216 |

*Table 1.* Number of Estimators - F1 Score Relationships

n_estimators give slightly better performance. However, the time grows exponentionally with n_estimators. So, it is not worth increasing n_estimators. In addition, we tried other parameters such as "min_samples_leaf, max_features, min_samples_split", however neither they improved the performance nor they ran fast. So, the main parameters that we can tune are "n_estimators" and "max_depth". The best model obtained from the table and plot above are used for final prediction for the competition, and the score is 0.7214.

### 4.5. Neural Network

Neural networks are widely used in machine learning problems. We trained neural networks for our problem under many different settings using the methods we proposed in the approach section.

Here, loss curves of the first neural network is shown in figure 7.

We used Early Stopping technique to obtain the best model in a given setting.The patience for early stopping was 20 in the experiments below.

As seen in the table, despite we tried many possibilities on "different layer dimensions, batch sizes, learning rates", but we could not reach the score of baseline. Then we decided to use fixed hidden dimensions such as [128,256,512,1024]

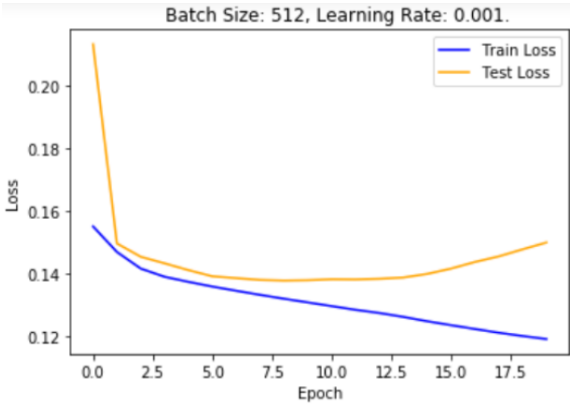


*Figure 7.* Neural Network Loss Curves

| Layer Dims. | LR | Batch Sz. | Acc. | f1 scr. |
|---|---|---|---|---|
| 256, 128 | .001 | 512 | 0.689 | 0.679 |
| 256, 128 | .001 | 1024 | 0.689 | 0.677 |
| 256, 128 | .0001 | 512 | 0.652 | 0.633 |
| 256, 128 | .0001 | 1024 | 0.641 | 0.622 |
| 256, 128, 64 | .001 | 512 | 0.694 | 0.685 |
| 256, 128, 64 | .001 | 1024 | 0.690 | 0.681 |
| 256, 128, 64 | .0001 | 512 | 0.669 | 0.654 |
| 256, 128, 64 | .0001 | 1024 | 0.660 | 0.640 |

*Table 2.* Accuracies and f1 scores of different Neural Networks.

and fixed the number of hidden layers as 5 and 10.We also fixed the batch size to 512. And we tried different optimization techniques and different loss functions under these settings. Since every change in the model, changes the optimal learning rate we chose the learning rate with fast.ai's learning rate finder. We started experimenting with hidden dimensions of 128 and layer number of 10 with the optimizer SGD with momentum and learning rate scheduler StepLR. By looking at figure 8 we observed that the learning rate reduces unnecessarily and we switched our learning rate scheduler to ReduceLR.

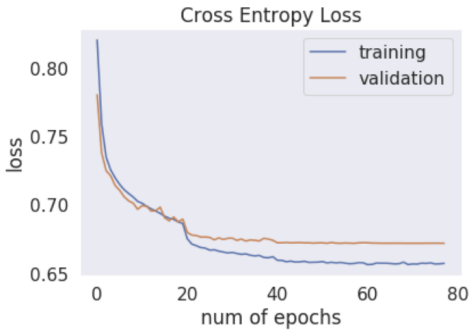


*Figure 8.* Loss-Epoch Relationship

After changing the learning scheduler we got better results as it can be seen from the results table. Then we tried different hidden dimensions since the results didn't improve enough and the model is computationally expensive, we ran the last experiments on with dimension of hidden layers is 128 and the number of layers is 10. By looking at one of the confusion matrices from our SGD + ReduceLR settings in figure x we can easily see that the class imbalance problem has a really bad impact on the training of neural networks.

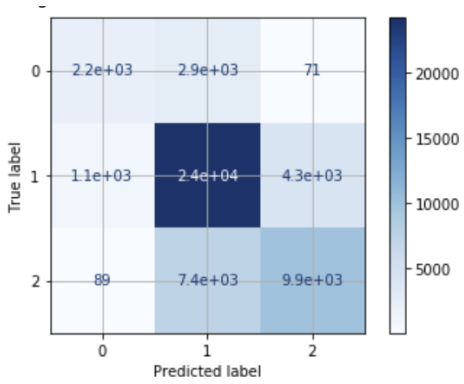


*Figure 9.* Confusion Matrix for SGD + ReduceLR strategy

Then we trained the neural networks with WCE loss and obtained better results. We also tried focal loss with gamma=1.2 but the focal loss didn't yield better results. The best result we obtained was an f1 score of 0.6931 with ADAM + ReduceLR strategy using WCE loss. By looking at confusion matrix of our best model at figure 10, we can say that it didnt reduce the class imbalance problem but somehow it got a better accuracy.We can obtain better scores by tuning the parameters and weights more but training neural networks are quite expensive and we can obtain better results with simpler models. We show that neural networks can help us to predict the damage level of a building but we can obtain better results with simpler models.

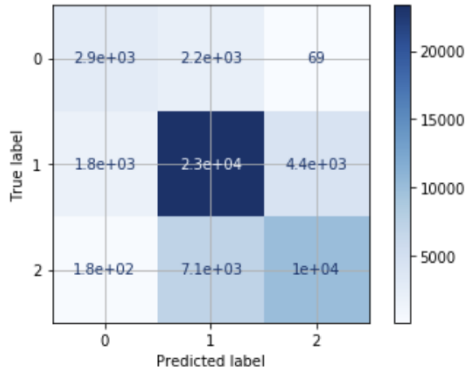


*Figure 10.* Confusion matrix for ADAM + ReduceLR strategy

| Dim. | num. layers | LR | Optim | Loss Function | Acc. | f1 scr. |
|---|---|---|---|---|---|---|
| 128 | 5 | 1.32e-1 | SGD + StepLR | CE | 0.6886 | 0.6798 |
| 128 | 10 | 8.71e-2 | SGD + StepLR | CE | 0.6823 | 0.6700 |
| 128 | 5 | 1.32e-1 | SGD + ReduceLR | CE | 0.6922 | 0.6843 |
| 128 | 10 | 8.71e-1 | SGD + ReduceLR | CE | 0.6964 | 0.6884 |
| 256 | 5 | 1.37e-1 | SGD + ReduceLR | CE | 0.6948 | 0.6876 |
| 256 | 10 | 1.32e-1 | SGD + ReduceLR | CE | 0.6950 | 0.6879 |
| 512 | 5 | 7.69e-2 | SGD + ReduceLR | CE | 0.6947 | 0.6881 |
| 512 | 10 | 1.03e-1 | SGD + ReduceLR | CE | 0.6939 | 0.6874 |
| 1024 | 5 | 1.56e-1 | SGD + ReduceLR | CE | 0.6950 | 0.6887 |
| 1024 | 10 | 1.16e-1 | SGD + ReduceLR | CE | 0.6924 | 0.6861 |
| 128 | 10 | 2.56e-1 | SGD + ReduceLR | WCE | 0.6931 | 0.6900 |
| 128 | 10 | 1.24e-1 | SGD + ReduceLR | FL | 0.6904 | 0.6820 |
| 128 | 10 | 2.56e-1 | SGD + WR | FL | 0.6812 | 0.6794 |
| 128 | 10 | 1.00e-3 | ADAM + ReduceLR | WCE | **0.7030** | **0.6931** |

*Table 3.* Accuracies and f1 scores of different Neural Networks.

### 4.6. Boosting: XGBoost

'Boosting' is a method that makes a strong learner from a collection of weak learners. Here weak learner stands for a model of less number of features-parameters. Weak learners make predictions of specific features where strong learners unite weak learners under one roof and at each step, the next weak learner takes into account the mistakes of the previous one. So, boosting is an ensemble method for improving the performance of any algorithm. Boosting algorithms generally do not overfit, since at every step it takes into account different weighted features. So after the unsatisfying results, we made our final shot with a boosting algorithm.

For boosting, we used the XGBoost method. It has a lot of parameters to tune and it is time-consuming to consider all. So, we took the most effective parameters and wanted to see their effects on results. In the table below, a small sample of these parameters and their results are shown. In one table we can not show all possible models, instead, we will show the tuning of our model step by step.

| Depth | LR | sbsmpl | csbyt | Gamma | f1 |
|---|---|---|---|---|---|
| 8 | 0.001 | 0.8 | 0.3 | 0 | 0.5979 |
| 8 | 0.001 | 0.8 | 0.3 | 1 | 0.5979 |
| 8 | 0.001 | 0.8 | 0.3 | 5 | 0.5979 |
| 8 | 0.001 | 0.8 | 0.5 | 0 | 0.6351 |
| 8 | 0.001 | 0.8 | 0.5 | 1 | 0.6351 |
| 8 | 0.001 | 0.8 | 0.5 | 5 | 0.6351 |
| 8 | 0.001 | 0.8 | 0.8 | 0 | 0.6607 |
| 8 | 0.001 | 0.8 | 0.8 | 1 | 0.6607 |
| 8 | 0.001 | 0.8 | 0.8 | 5 | 0.6607 |

*Table 4.* F1 Scores of different parameter combinations, 3-CV.

The parameters that we considered are "max_depth", "learning rate", "subsample", "colsample_bytree" and "gamma". Here are the brief explanations of these parameters:

- **max_depth:** Maximum depths of trees used.
- **learning_rate:** Effects the change of steps in gradient descent.
- **subsample:** Subsample ratio of the training samples.
- **colsample_bytree:** Ratio of the used features when constructing the trees.
- **gamma:** The minimum value of loss decrease to make a partition on the leaves.

In the table, it can be seen that the 'gamma' value never changes the score. So we can ignore it in our next experiments. Besides, higher "colsample_bytree" gave a higher score. In the next step, the optimum "max. depth" is found.

Here, a sample plot for "Max. Depth of GBTree - F1 Score" is shown in figure.

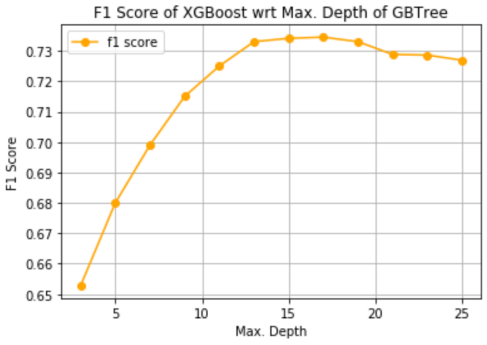


*Figure 11.* F1 Score - Depth of GBTree on XGBoost

As seen in the figure, the model was able to perform best when "max. depth = 17", with 0.7380 f1 score on this dataset. This result is satisfying, especially when considering the leaderboard of the competition the 1st place had f1 score of 0.7558 -. At this point, the next objective is to optimize this model.

So we will also discard the lowest value of that parameter and here is the corresponding f1 score table.

| LR | Subsample | cs_bytree | f1 score |
|---|---|---|---|
| 0.001 | 0.8 | 0.5 | 0.7087 |
| 0.001 | 0.8 | 0.8 | 0.7299 |
| 0.001 | 1.0 | 0.5 | 0.7105 |
| 0.001 | 1.0 | 0.8 | 0.7294 |
| 0.01 | 0.8 | 0.5 | 0.7200 |
| 0.01 | 0.8 | 0.8 | 0.7332 |
| 0.01 | 1.0 | 0.5 | 0.7209 |
| 0.01 | 1.0 | 0.8 | 0.7335 |
| 0.1 | 0.8 | 0.5 | 0.7394 |
| 0.1 | 0.8 | 0.8 | 0.7385 |
| 0.1 | 1.0 | 0.5 | **0.7424** |
| 0.1 | 1.0 | 0.8 | 0.7382 |

*Table 5.* F1 Scores of different parameter combinations, 3-CV.

In the table above, it can be seen that a higher "learning rate" has better performance. Besides, when the "learning rate" values are kept constant, in general, higher "subsample" and "colsample_bytree" values gave better performance. With 3-Cross Validation, the best model is the one which has "0.1 learning rate, 1.0 subsample, 0.5 colsample_bytree" with 0.7424 f1 score.

With this saved model, when we predict the "Test Data" on the competition which can be found here (LINK), the score was 0,7456. This is quite satisfying because the highest score was 0,7558.

there is a collection of steps we followed: we tried focal loss and sampling methods which we mentioned earlier to see if they increase the performance. The step here is to re-run the previous algorithms with the sampled data and focal loss. In the table below, the before-after f1 scores of the algorithms are shown.

## 5. Conclusion

We applied 6 machine learning methods and analyzed Random Forest, Neural Networks and XGBoost in a very deep fashion.We tried different depths,number of estimators,min samples leaf, max features,min samples split parameters for Random Forest and analyzed their affects on this task and the best score we obtained was f1 score of 0.7214 with Random Forest algorithm. We tried different training techniques and different models for neural networks and the best score we obtained was 0.6931.Since the data set is quite noisy and imbalanced neural networks were not as effective as we expected.And last of all we tried XGBoost with many different parameters and analyzed their effects. The best score we obtained with XGBoost was f1 score of 0.7424 on the test set, and we got f1 score of 0.7454 on the challange, where the best result is currently 0.7558.You can see all the results we got with the algorithms we used in table 6.We show that we can use machine learning models to predict damage level of a building by using it's features.

| Algorithm | f1 score) |
|---|---|
| K-NN | 0.7030 |
| Weighted K-NN | 0.7073 |
| Naive Bayes | 0.3410 |
| Logistic Regression | 0.4923 |
| Random Forest | 0.7214 |
| Neural Networks | 0.6931 |
| XGBoost | 0.7424 |

*Table 6.* Results for all algorithms used

## 6. Future Work

Throughout this project, we discovered the effects of fundamental Machine Learning algorithms on our dataset and finally, reached an acceptable performance with boosting methods.In future work we plan to clean this data from outliers and apply feature selection on the dataset. We also plan to train a better neural network by tuning the parameters and playing with the data.And last of all we plan to collect a new dataset from the government records of Big Istanbul Earthquake (1999) and apply the same methods on this dataset.

## References

[1] Earthquake damage modeling dataset.

[2] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006.

[3] Jeremy Howard et al. fastai. https://github.com/fastai/fastai, 2018.

[4] Hanxi Jia, Junqi Lin, and Jinlong Liu. An earthquake fatalities assessment method based on feature importance with deep learning and random forest models. *Sustainability*, 11(10):2727, 2019.

[5] Amin Karbassi, Benyamin Mohebi, Shaliz Rezaee, and Pierino Lestuzzi. Damage prediction for regular reinforced concrete buildings using the decision tree algorithm. *Computers & Structures*, 130:46–56, 2014.

[6] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.

[7] T.-Y. Lin, P. Goyal, R. B. Girshick, K. He, and P. Dollár. Focal loss for dense object detection. In *ICCV*, pages 2999–3007, 2017.

[8] I. Loshchilov and F. Hutter. SGDR: Stochastic gradient descent with warm restarts, 2017.

[9] Samuel Roeslin, Quincy Ma, Ken Elwood, and Jorg Wicker. Development of a seismic damage prediction model using machine learning. 2018.

[10] Naito Shohei, H Nakamura, H Fujiwara, M Naoichi, and T Hiromitsu. The development of damage identification methods for buildings with image recognition and machine learning techniques utilizing aerial photographs of the 2016 kumamoto earthquake. In *AGU Fall Meeting Abstracts*, 2017.

[11] PS Skjaerbaek, Søren RK Nielsen, Poul Henning Kirkegaard, and AŞ Çakmak. Damage localization and quantification of earthquake excited rc-frames. *Earthquake engineering & structural dynamics*, 27(9):903–916, 1998.

[12] I. Sutskever, J. Martens, G. Dahl, and G. Hinton. On the importance of initialization and momentum in deep learning. In *ICML*, 2013.

[13] David Vere-Jones. Forecasting earthquakes and earthquake risk. *International Journal of Forecasting*, 11(4):503–538, 1995.

[14] Yang Yucheng, Yang Liu, Gao Yunxue, Yang Yaling, Lu Xilei, and Yang Guizhen. Method of damage prediction for existing multi story brick buildings and its reliability [j]. *Earthquake Engineering and Engineering Vibration*, 3, 1982.