# Modeling Earthquake Damage

Capstone Project: Samantha Werdel

## Abstract

Natural disasters are unpreventable, we can try and predict when the disasters will happen but can't stop them. In this report, instead of predicting a natural disaster, I will predict the aftermath of the destruction from the structure of buildings. The ability to predict the level of damage caused by a natural disaster would be beneficial in finding what building structures sustain minimal damage. The dataset contains information about a building's architect and history that was hit by the Nepal earthquake in 2015. In this report, I aim to predict the level of destruction caused by the earthquake. With this classification problem I will use keras' deep learning neural network to fit a model and predict the level of damage because the ability of deep learning to incrementally learn the data a pattern would give more accuracy with prediction. However, after extensive tuning of the keras model, I found that even when using a deep learning network, sometimes the simpler model is better.

## Introduction

The Data Driven competition for predicting earthquake damage has over 600 competitors. While researching possible approaches to take, there have been many published reports and code on how to approach this problem. Many, like myself, use a machine learning model to solve this problem. Some took a more mathematical approach. Other earthquake damage modeling has been done outside of the competition dataset. I approach this problem with the knowledge of other models that have been attempted to understand the features and how to use them to forecast the outcome.

## Dataset

The dataset comes from the Driven Data competition, Richter's Predictor: Modeling Earthquake Damage [1]. The data was collected through surveys by the Central Bureau of Statistics after the 2015 earthquake in Nepal and is one of the largest post-disaster datasets ever collected. The data consists of three datasets: train values, train labels, and test values. The train and test values comprise of 260,601 and 86,868 rows respectively and 39 variables. These variables are mainly comprised of the buildings' structure. Both values and labels datasets contain the buildings' id, and the train labels dataset also include the damage grade for that building. The damage grade is represented by 3 levels; 1 represents low damage, 2 represents a medium amount of damage, and 3 represents almost complete destruction.

The datasets were relatively clean. After collecting the data and the compilation of the respective datasets into a data frame, exploratory analysis confirmed that there were no missing values and each variable contained the correct corresponding datatypes. The variables in the train values dataset contained both numeric and categorical data. To get consistent datatypes of the features, the construction of dummy variables from the categorical data were done. This increased the

| | building_id | geo_level_1_id | geo_level_2_id | geo_level_3_id | count_floors_pre_eq | age | area_percentage |
|---|---|---|---|---|---|---|---|
| 1 | 802906 | 6 | 487 | 12198 | 2 | 30 | 6 |
| 2 | 28830 | 8 | 900 | 2812 | 2 | 10 | 8 |
| 3 | 94947 | 21 | 363 | 8973 | 2 | 10 | 5 |
| 4 | 590882 | 22 | 418 | 10694 | 2 | 10 | 6 |
| 5 | 201944 | 11 | 131 | 1488 | 3 | 30 | 8 |

| | building_id | damage_grade |
|---|---|---|
| 1 | 802906 | 3 |
| 2 | 28830 | 2 |
| 3 | 94947 | 3 |
| 4 | 590882 | 2 |
| 5 | 201944 | 3 |

**Figure 1.** Sample data from the train values dataset(left) and train labels dataset(right).

number of features to 61, including the building id. With this number of features, a machine learning technique would need to be implemented to learn the pattern of the data to make the most accurate predictions.

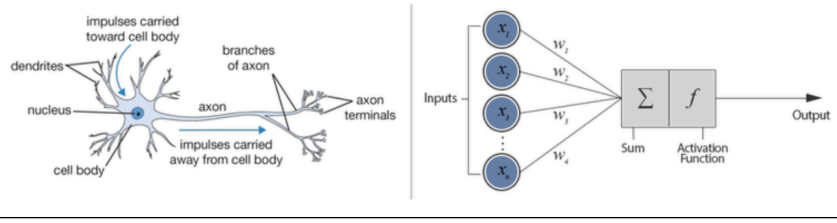



**Figure 2.** Representative image of the neuron vs deep learning [3].

## Proposed Method

With the many different types of machine learning models to choose from and after running initial models from the sklearn library. I wanted to use a more advance machine leaning model which involved deep learning. Deep learning uses artificial neural networks, a term which was motivated by our own neural network in the human brain. Neurons receive signals, if that signal is strong enough, will prompt the neuron to fire its signal to other neurons. Similar to the signal a neuron receives, in deep learning, the input data is the signal. In deep learning, how to determine what input data to send to the next layer is assigned by weights of importance. A deep learning model can have as many layers. Increasing the layers increases the complexity of the model, but you must watch out for overfitting.

The deep learning model I will use throughout the report is keras. The specific model I use is Sequential, which models a linear stack of layers, and the specific layer I use is Dense, a densely-connected neural network [2]. The first layer is for the input data, which takes the size of the features from the data used to predict the output. And the last layer is your output. The layer(s) in-between is variable. Included in these layers is the activation function, determines if the "signal is strong enough". The compilation to train the data has many parameters. For my model, I focused on three; loss function, optimizer, metric. The loss function measures how will the model is doing. The optimizer is a mechanism that allows the model to update itself based on the input seen and how the loss function is preforming. And the metric to evaluate the model during training and testing. With much tuning and trial and error of the model, Figure 3 shows the model that returns the best results.

```
# final model
model = Sequential()
model.add(Dense(128, input_dim=X_train.shape[1], activation='sigmoid'))
model.add(Dense(units=128, activation=ReLU(max_value=None, negative_slope=0.0, threshold=0.0)))
model.add(Dropout(0.1))
model.add(Dense(units=128, activation=ReLU(max_value=None, negative_slope=0.0, threshold=0.0)))
model.add(Dropout(0.1))
model.add(Dense(units=128, activation=ReLU(max_value=None, negative_slope=0.0, threshold=0.0)))
model.add(Dropout(0.1))
model.add(Dense(3, activation='softmax'))

# Compile model
model.compile(loss='categorical_crossentropy', optimizer='adamax', metrics=['accuracy'])

lr_call = lr_mon()
lr_scheduler=keras.callbacks.ReduceLROnPlateau(monitor='val_acc', factor=0.1, patience=10, verbose=0,
                                               mode='auto', min_delta=0.0001, cooldown=0, min_lr=0)

history = model.fit(X_train,y_train, batch_size=64, epochs=80, validation_data=(X_val,y_val),
                    callbacks=[lr_call,lr_scheduler])
```

**Figure 3.** Final keras sequential model with 4 layers, with loss function as categorical crossentropy and optimizer is adamax.

## Experiment Detail

### *Exploratory Data Analysis*

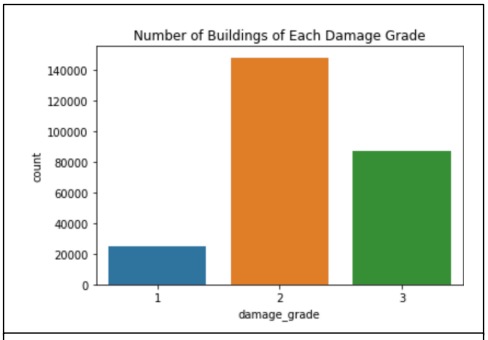
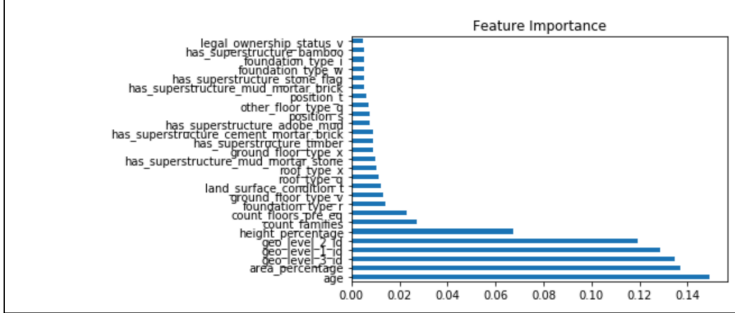



**Figure 4.** Distribution of train labels.

First, before delving into the features of the data, I wanted to look at the distribution of the train labels. Figure 4 shows the count of each damage level. The damage grade at level 2 is noticeably of the outcomes, with more than 50%. I next want to explore the features before any modification, dummy variables, are done. Exploring the numerical data first, I see that the ranges of the features differ. Some features are id, which goes up to 12,567, this is a drastic number compared to the count of floors which only goes up to nine. The distribution of these features is normal, with the exception of extreme outliers. These concerns I address more on in the *preprocessing data* section. Next, I explore the categorical features in the dataset. There seems to be a major disparity within each categorical feature where one value holds the majority. Figure 5 shows a clear difference in roof type n and the other types of roofing. And the foundation type feature shows an even greater gap were type r composes of nearly all the foundation being used and type h has virtually none. The *processing data* section will take into account these distributions.

Looking at the data as a whole again. Before modeling, I want to see if there is some pattern with the data I can detect. Figure 5 shows the feature importance of the variables that have importance of at or above 0.05%. There does not seem to be any leading features. Age and area percentage are the highest but only at 15% and 13%. This does make the most sense because the sturdiness and structure of a building can depend on how old it is and also with the area of the building, a bigger building could withstand an earthquake better than a smaller one. Even so, with little feature importance a deep learning model is advantageous in being able to detect an underlying pattern within the dataset.

| | geo_level_1_id | geo_level_2_id | geo_level_3_id | count_floors_pre_eq | age | area_percentage | height_percentage |
|---|---|---|---|---|---|---|---|
| count | 260601.000000 | 260601.000000 | 260601.000000 | 260601.000000 | 260601.000000 | 260601.000000 | 260601.000000 |
| mean | 13.900353 | 701.074685 | 6257.876148 | 2.129723 | 26.535029 | 8.018051 | 5.434365 |
| std | 8.033617 | 412.710734 | 3646.369645 | 0.727665 | 73.565937 | 4.392231 | 1.918418 |
| min | 0.000000 | 0.000000 | 0.000000 | 1.000000 | 0.000000 | 1.000000 | 2.000000 |
| 25% | 7.000000 | 350.000000 | 3073.000000 | 2.000000 | 10.000000 | 5.000000 | 4.000000 |
| 50% | 12.000000 | 702.000000 | 6270.000000 | 2.000000 | 15.000000 | 7.000000 | 5.000000 |
| 75% | 21.000000 | 1050.000000 | 9412.000000 | 2.000000 | 30.000000 | 9.000000 | 6.000000 |
| max | 30.000000 | 1427.000000 | 12567.000000 | 9.000000 | 995.000000 | 100.000000 | 32.000000 |



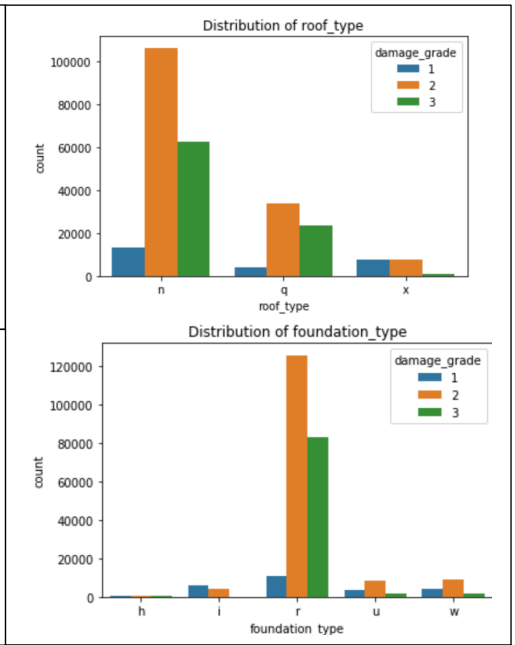




**Figure 5.** Statistical summary of numerical features (top left). Top variables of feature importance (bottom left). Distribution count of categorical features (right). Roof type (top) Foundation type (bottom).

### *Primary Model*

Before exploring the deep learning model, I used models in the sklearn library from my previous experiences to perform on the data. After running a few classifier models, I found BaggingClassifier produced the best performance. With this model as my benchmark, I worked in keras to construct the best model.

### *Preprocessing Data*

The disparity in the features presents issues when modeling the data. Therefore, before I input the data into the model, I first standardized the features in the data. I interpreted the train labels of damage grade as categorical, thus a one-hot encoding was applied to transform the labels into three classes.

The objective of the competition is to predict the damage grade of the buildings in the test values dataset. Because the test labels were not given, in order to build the predictive model, the training values and labels were split into train and validate datasets. The training data was split 80/20 to randomly select the training and validation dataset.

### *Results*

To test how well the model predicted the damage grade of the test data, the predicted test labels are submitted to the competition. The competition uses an f1_score to determine how well a model performed. The f1 performance score I received was 0.697. This score was lower than the initial BaggingClassifier model, which generated a score of 0.711. Figure 6 shows how the keras model learned and performed over the time it was learning and fitting the data. While loss continuously decreases in the training dataset to about 0.64, the validation loss plateaus at around 0.67. Similar with the accuracy, the training data continuously increases to about 0.71, and looks like it could continuously increase, whereas the validation dataset, not as noticeably but does levels out around 0.69.

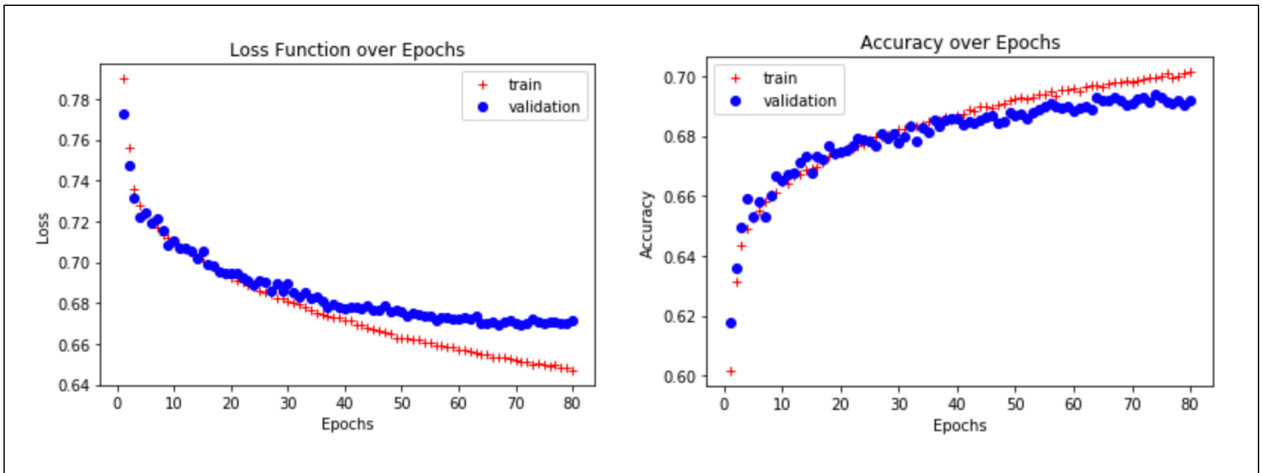



**Figure 6.** Visualization of model while learning

## Error

The visualizations from the keras model's performance demonstrates a possible over-fitting of the training model. The plateau in both the accuracy and loss of the validation is reason of that. This possible over-fitting could be caused by too many layers in the model.

## Conclusion

It seemed that even after persistent tuning and training with a deep learning model, the first simpler model was the best. I was curious to see if I could further improve the primary BaggingClassifier model by hyper-tuning two of the parameters. I used GridSearchCV to find the best parameters to implement on the data and was able to get a considerably better score of 0.746. Currently ranking 30$^{th}$ in the competition. I would be interested to see if with more parameter tuning I could bet that current score.

## Citations


1. DrivenData. (n.d.). Richter's Predictor: Modeling Earthquake Damage. Retrieved from https://www.drivendata.org/competitions/57/nepal-earthquake/page/134/
2. The Sequential model API. (n.d.). Retrieved from https://keras.io/models/sequential/
3. Towards Data Science. (2017, June 19). From Fiction to Reality: A Beginner's Guide to Artificial Neural Networks. Retrieved from https://towardsdatascience.com/from-fiction-to-reality-a-beginners-guide-to-artificial-neural-networks-d0411777571b
4. Deep learning. (2019, July 24). Retrieved from https://en.wikipedia.org/wiki/Deep_learning