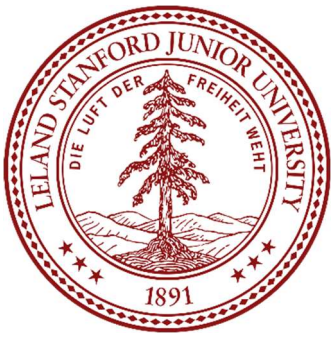

STANFORD UNIVERSITY

CS 229 – Spring 2021

Machine Learning

Final Project Report

Application of Machine Learning Methods to Predict the Level of Buildings Damage in the Nepal Earthquake

Yitian Liang
(SUNetID: ytliang)

June 2, 2021

1 Abstract

The assessment of damage level of post-earthquake structures through calculations is always complicated and time-consuming, and it is not suitable for situations when a large number of buildings need to be evaluated after an earthquake. One of the largest post-disaster datasets, which comes from the April 2015 Nepal earthquake, makes it possible to train models to predict damage levels based on building information. Here, we investigate the application of various machine learning methods to the problem of structural damage level prediction.

2 Introduction

The ground movements in earthquakes will cause huge damage to building structures. There are different levels of earthquake magnitude, but there is no systematic classification of the buildings damage. The dynamic analysis of a single structure usually takes a long time, which is not suitable for situations where a large number of buildings need to be evaluated after an earthquake. After the April 2015 Nepal earthquake, which was the worst natural disaster to strike Nepal since 1934, the National Planning Commission, along with Kathmandu Living Labs and the Central Bureau of Statistics, has generated large post-disaster datasets, containing valuable information on earthquake impacts, household conditions, and socio-economic-demographic statistics. I hope that through the application of machine learning on this dataset, we can quickly classify the damage level of the structures in order to facilitate planning rescue and reconstruction and assessing the loss.

I was inspired by an ongoing open competition, called Richter's Predictor: Modeling Earthquake Damage, on DrivenData. The input to our algorithm is a collect of information on post-disaster buildings. We then use K-Nearest Neighbors, Backpropagation Neural Network and Random Forests to output a predicted level of damage for each building.

3 Related work

The study of earthquake damage to buildings began in the 18th century and has a long history. One of the first papers to explore the damage of earthquake was Rasita Kerdpoln (1787), who discussed vulnerability to natural disasters as an issue of power based on earthquake in Haiti. For more than one hundred years, scholars in the civil engineering field have been studying the influence of various factors on damage of post-earthquake structures. In 1984, A.C. Boissonnade et al. presented a consistent method for earthquake intensity classification based on the theory of statistical pattern recognition and developed a discriminative function for such identifications based on the Bayesian criterion. Until 2020, Gian P. Delacruz used Generative Adversarial Networks to classify structural damage caused by earthquakes, which is almost the earliest application of machine learning in classifying damage of post-earthquake structures. Samuel Roeslin et al. (2020) tried four algorithms to develop a damage prediction model from 340 post-earthquake buildings in the Mexico City and achieved more than 65% prediction accuracy. Similarly, Shohei Naito et al. (2020) used machine learning models and aerial photographs to classify buildings in the Kumamoto earthquake into four damage levels. In this project, we will be expanding on these works and applying more varied methods on a much larger dataset, trying to achieve higher prediction accuracy.

4 Dataset and Features

The dataset, which comes from the 2015 Nepal Earthquake Open Data Portal, is one of the largest post-disaster datasets ever collected. It has already been divided into three parts:
(1) In the ***train_values.csv***, each row represents a specific building in the region that was hit by Gorkha earthquake, and there are 260601 examples in total. Each building is identified by a unique *building_id*, which can be used as an index. There are 38 features, including structural information such as the number of floors (before the earthquake), age of the building, and type of foundation, as well as legal information such as ownership status, building use, and the number of families who live there. The data types of the features are shown as below:

geo_level_1_id	int64	has_superstructure_cement_mortar_brick	int64
geo_level_2_id	int64	has_superstructure_timber	int64
geo_level_3_id	int64	has_superstructure_bamboo	int64
count_floors_pre_eq	int64	has_superstructure_rc_non_engineered	int64
age	int64	has_superstructure_rc_engineered	int64
area_percentage	int64	has_superstructure_other	int64
height_percentage	int64	legal_ownership_status	object
land_surface_condition	object	count_families	int64
foundation_type	object	has_secondary_use	int64
roof_type	object	has_secondary_use_agriculture	int64
ground_floor_type	object	has_secondary_use_hotel	int64
other_floor_type	object	has_secondary_use_rental	int64
position	object	has_secondary_use_institution	int64
plan_configuration	object	has_secondary_use_school	int64
has_superstructure_adobe_mud	int64	has_secondary_use_industry	int64
has_superstructure_mud_mortar_stone	int64	has_secondary_use_health_post	int64
has_superstructure_stone_flag	int64	has_secondary_use_gov_office	int64
has_superstructure_cement_mortar_stone	int64	has_secondary_use_use_police	int64
has_superstructure_mud_mortar_brick	int64	has_secondary_use_other	int64

Figure 1: Data types of the 38 features

(2) In the ***train_labels.csv***, there are 2 columns and 260601 rows. Every *building_id* in the training values data has a corresponding label in this file. A *1* represents low damage, a *2* represents a medium amount of damage, and a *3* represents almost complete destruction.
(3) In the ***test_values.csv***, these are the features of 86868 buildings. I will use this set to make predictions after training a model to evaluate performance. The labels of the test set are not given and I need to submit my predictions to the competition on DrivenData to get the accuracy.

5 Methods

5.1 K-Nearest Neighbors

K-Nearest Neighbors is supervised machine learning algorithm which can be used for both classification and regression problems. KNN classification is to classify an unknown object from known objects. It uses standard Euclidean distance to measure the variation between the training instance and test instance. The standard Euclidean distance $d(x_i, x_j)$ is defined below as [1] :

$$d(x_i, x_j) = \sqrt{\sum (a_r(x_i) - a_r(x_j))^2}$$

KNN computes the most common class of K nearest neighbors to estimate the class of the test instance of the test set. It is defined below as [1]:

$$c(x) = \arg\max \sum_{i=1\ to\ K} d(c, c(y_i))$$

The reason for selecting KNN classifier is its simplicity, efficiency and effectiveness. It is one of

the most simple and fundamental classification methods. In real world scenario, there are many data sets of which there is little or no prior knowledge about its distribution. To deal with such data sets KNN is amongst the best choice for classification [2].

5.2 Neural Network

Neural networks refer to broad type of non-linear models/parametrizations $h_\theta(x)$ that involve combinations of matrix multiplications and other entrywise non-linear operations [3]. For a simple two-layer neural network used in this project, the algorithm is shown below as:

$$z^{[1]} = W^{[1]}x + b^{[1]}$$
$$a = sigmoid(z)$$
$$z^{[2]} = W^{[2]}a + b^{[2]}$$
$$h_\theta(x) = softmax(z^{[2]})$$

For this problem, we will use a regularized backpropagation-trained neural network (BPNN) based on gradient descent. In this case, the cost function is defined as follows

$$J = \left(-\frac{1}{B}\sum_{i=1}^{B}\sum_{k=1}^{K} y_k^{(i)} \log \hat{y}_k^{(i)}\right) + \lambda(\left|\left|W^{[1]}\right|\right|^2 + \left|\left|W^{[2]}\right|\right|^2)$$

The reason for choosing BPNN is [4]: (1) Backpropagation is fast, simple and easy to program; (2) It has no parameters to tune apart from the numbers of input; (3) It is a flexible method as it does not require prior knowledge about the network; (4) It is a standard method that generally works well; (5) It does not need any special mention of the features of the function to be learned.

5.3 Random Forests

Random forests (random decision forests) are an ensemble learning method for classification, regression and other tasks that operates by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean/average prediction (regression) of the individual trees.[5][6] The Working process can be explained in the below steps [7]: (1) Select random K data points from the training set; (2) Build the decision trees associated with the selected data points (Subsets); (3) Choose the number N for decision trees that you want to build; (4) Repeat (1) &(2); (5) For new data points, find the predictions of each decision tree, and assign the new data points to the category that wins the majority votes. When performing Random Forests based on classification data, we often use the Gini index to decide how nodes on a decision tree branch [8]:

$$Gini = 1 - \sum_{i=1}^{c}(p_i)^2$$

where p_i represents the relative frequency of the class we are observing in the dataset and c represents the number of classes.

The reasons why I choose the random forests are as follows [9]:

(1) Simplicity. Random Forests require almost no input preparation and the basic RF learning algorithm can be written in a few lines of code with lots of open-source implementations.

(2) Good performance. Random forests are not very sensitive to the specific hyper-parameters used. They don't require a lot of tweaking and fiddling to get a decent model and they are very quick to train when there are a large number of examples and features.

6 Experiments and Results

6.1 K-Nearest Neighbors

K-Nearest Neighbors (KNN) algorithm is a simple but lazy learner, because it doesn't learn a discriminative function from the training data but need to search for the nearest neighbor for each prediction. To reduce the training time and make KNN less expensive, we use *sklearn.feature_selection.SelectKBest* to select the 20 most important features. The SelectKBest class scores the features using a function *f_classif* and chooses the 20 highest scoring features as shown below:

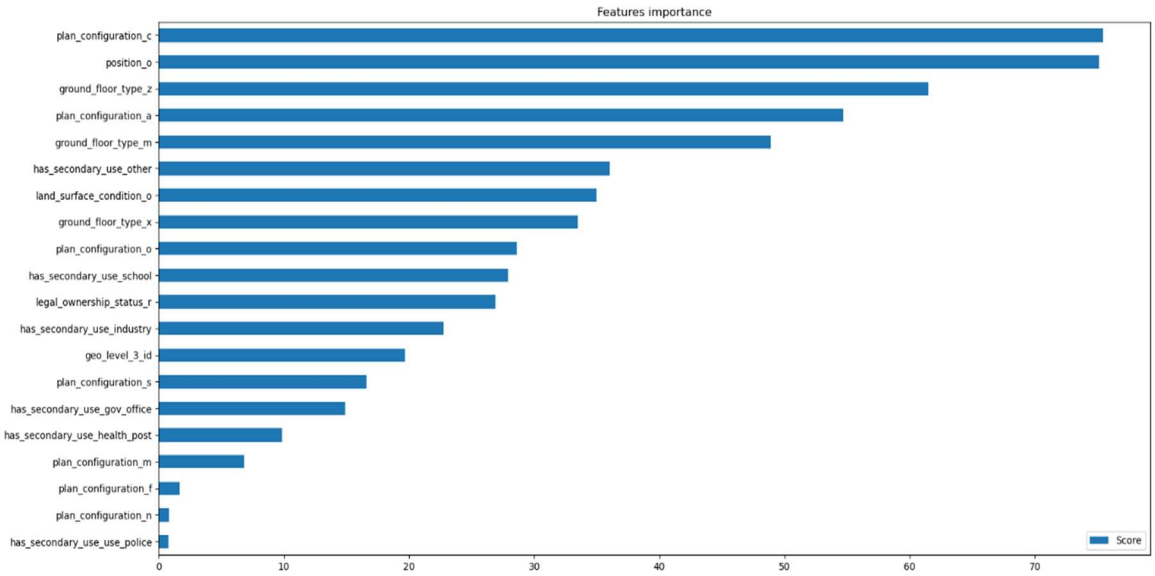

Figure 2: The 20 most important features and scores

We know from 5.1 that the distance calculation in KNN uses feature values. When one feature values are larger than other, that feature will dominate the distance and the outcome of the KNN. Therefore, we use *sklearn.preprocessing.StandardScaler* to standardize features by removing the mean and scaling to unit variance.

To implement KNN, we use *sklearn.neighbors.KNeighborsClassifier* to fit the train values and train labels, and predict the test set.

The evaluation metric used for this project is the micro-averaged F1 score, which is great way to assess the quality of multi-label binary problems. The micro-averaged F1 score is defined as below:

$$F_{micro} = \frac{2P_{micro}R_{micro}}{P_{micro} + R_{micro}}$$

where P_{micro} is the micro precision, and R_{micro} is the micro recall.

When we set the parameter $\mathrm{n\_neighbors} = 7$, the micro-averaged F1 score is **0.6488**.

6.2 Neural Network

For this project, we use a two-layer regularized backpropagation neural network with standardized data. The number of input neurons is 38 and the number of output neurons is 3, because we have 38 features and 3 classes. The number of hidden neurons and the learning rate can be approximately defined as below [10]:

$$N_h = \sqrt{N_i N_o} \qquad \alpha = 32/N_h$$

where N_h - number of hidden neurons, N_i - number of input neurons, N_o - number of output

neurons, α- learning rate.

Therefore, we set $N_h = 10$, $\alpha = 5$ and $\lambda = 0.0001$ at the beginning. To accelerate the program with large number of examples, we set the mini batch size $B = 1000$. After using *sklearn.preprocessing.OneHotEncoder* to encode training labels as a one-hot numeric array, we run the neural network and find that the accuracy oscillate a lot with iterations, which suggests that the learning rate is too large and it causes undesirable divergent behavior in the loss function. We decrease the learning rate gradually and find that when α is approximately smaller than 3, the accuracy curve converges and stops oscillating.

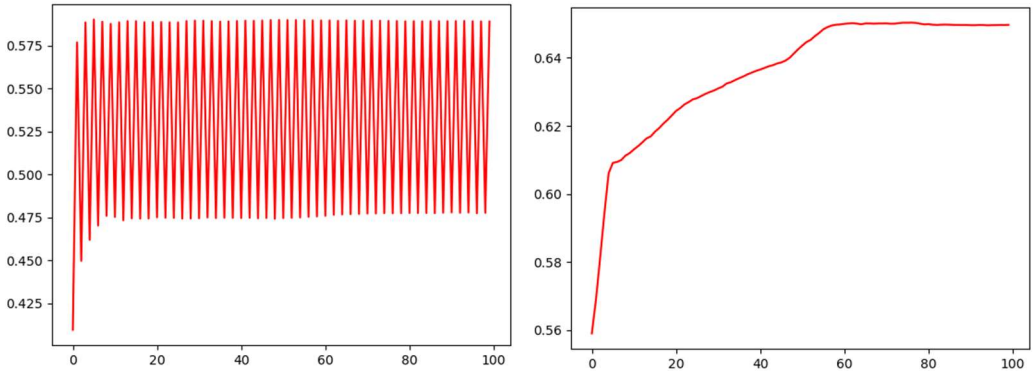

Figure 3: The change of accuracy with iterations when $\alpha = 5$ (left) and $\alpha = 2$ (right)

Since the randomly initialized weight will affect the results, we train the model for multiple times and the highest micro-averaged F1 score we get is **0.6502**.

6.3 Random Forests

Since Random Forests are a tree-based model and does not require feature scaling, we use the original dataset with 38 features. We assemble all the features into a pipeline and use the *make_pipeline* function to automatically name the steps in the pipeline as a lowercase version of whatever the object name is. Then we test a few different models using *GridSearchCV* with params_grid set as *{'min_samples_leaf': [1, 5], 'n_estimators': [50, 100]}* and get the best params as: *{'min_samples_leaf': 1, 'n_estimators': 100}*.

The in-sample micro F1 score for the training set is **0.9843.** Since a perfect micro F1 score would be 1, this is a relatively good try. Then we use the trained model to predict the test set and get a micro F1 score of **0.7150** for the test set.

7 Conclusion

Among the three algorithms in this project, in terms of computational efficiency, Neural Network is the highest and K-Nearest Neighbors is the lowest. With respect to the prediction accuracy, Random Forests are the highest and K-Nearest Neighbors is the lowest.

KNN algorithm usually works better with a small number of input variables, but struggles with high dimensions and large datasets. Our choice of the 20 most important features may further reduce the accuracy of the model to some extent. Neural Network algorithm runs fast on large high-dimensional datasets but the accuracy here is not high, probably because its structure and parameters have not been adjusted to the optimal combination since NN is relatively hard to train. From the results of Random Forests algorithm, we can see that the micro F1 score of training set is very close to 1 while that of test set is much lower, which is a sign of overfitting. Since random forests are prone to overfitting, it might help to have more trees or reduce the max depth of models.

8 References

1. S. Taneja, C. Gupta, S. Aggarwal and V. Jindal, "MFZ-KNN — A modified fuzzy based K nearest neighbor algorithm," 2015 International Conference on Cognitive Computing and Information Processing(CCIP), 2015, pp. 1-5, doi: 10.1109/CCIP.2015.7100689.
2. H. Parvin, H. Alizadeh and B. Minati, "A Modification on K-Nearest Neighbor Classifier", Global Journal of Computer Science and Technology, vol. 10, pp. 37-41, 2010.
3. R. Anand, K. G. Mehrotra, C. K. Mohan and S. Ranka, "An improved algorithm for neural network classification of imbalanced training sets," in IEEE Transactions on Neural Networks, vol. 4, no. 6, pp. 962-969, Nov. 1993, doi: 10.1109/72.286891.
4. Back Propagation Neural Network: Explained With Simple Example. https://www.guru99.com/backpropogation-neural-network.html#4
5. Ho, Tin Kam (1995). Random Decision Forests (PDF). Proceedings of the 3rd International Conference on Document Analysis and Recognition, Montreal, QC, 14–16 August 1995. pp. 278–282. Archived from the original (PDF) on 17 April 2016. Retrieved 5 June 2016.
6. Ho TK (1998). "The Random Subspace Method for Constructing Decision Forests" (PDF). IEEE Transactions on Pattern Analysis and Machine Intelligence. 20 (8): 832–844. doi:10.1109/34.709601.
7. Random Forest Algorithm. https://www.javatpoint.com/machine-learning-random-forest-algorithm
8. Madison Schott. Random Forest Algorithm for Machine Learning. https://medium.com/capital-one-tech/random-forest-algorithm-for-machine-learning-c4b2c8cc9feb
9. Ahmed El Deeb (2015). “The Unreasonable Effectiveness of Random Forests”.
10. Madhiarasan, M., Deepa, S.N. A novel criterion to select hidden neuron numbers in improved back propagation networks for wind speed forecasting. Appl Intell 44, 878–893 (2016). https://doi.org/10.1007/s10489-015-0737-z.