Analysis of Nepal Damage Grade

April 2018

Executive Summary

Our goal in this project is to predict the Nepal damage grade based on the 39 explanatory variables provided. This project gives me great opportunity to consolidate my ability in both data analysis and data prediction. I applied concepts and data analytic skills learned from the previous classes included Essential Statistics for Data Analysis using Excel, Data Science Essential, Introduction to Python for Data Science, Principles of Machine Learning, Programming with Python for Data Science. I used python as the tool to solve problems for the Nepal damage grade prediction project.

We are asked to predict the damage grade for each Building recorded undergo earthquake. As general, I will follow the path through out the project: data exploration, feature determining, data wrangling, along with data manipulation and visualization. Since the project is a classification problem, I will use machine-learning algorithm of SVM, KNN, Naïve Bayes, Random Forest, gradient boosting, XGboost as candidate models, then compare the accuracies out of them.

Exploring Damage in Nepal dataset

By using the info and describe function to get a general understanding of the data in python, we can see the training set has 10000 samples and 39 features, included 2 floats, 29 integer, and 8 objects.

```
1 train_value.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 39 columns):
.
.
.
dtypes: float64(2), int64(29), object(8)
```

The result below shows the description of the combine data of train value and train label, The labeled dataset has a mean of 2.2488 in 'damage_grade', we can tell the damage of earthquake to Nepal is really bad.

```
1 result = pd.merge(train_value, train_label)
2 result.describe()
```

	building_id	geo_level_1_id	geo_level_2_id	geo_level_3_id	count_floors_pre_eq	age	area	height	foundation_type	plan_config
count	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	0.0	
mean	9987.160000	7.135600	296.930300	2678.617900	2.146700	25.393500	38.438100	4.653100	NaN	
std	5800.800829	6.225567	279.390651	2520.663769	0.736365	64.482893	21.265883	1.792842	NaN	
min	1.000000	0.000000	0.000000	0.000000	1.000000	0.000000	6.000000	1.000000	NaN	
25%	4998.750000	2.000000	60.000000	606.750000	2.000000	10.000000	26.000000	4.000000	NaN	
50%	9963.500000	6.000000	219.000000	1937.500000	2.000000	15.000000	34.000000	5.000000	NaN	
75%	15044.750000	10.000000	466.000000	4158.000000	3.000000	30.000000	44.000000	5.000000	NaN	
max	19999.000000	30.000000	1411.000000	12151.000000	9.000000	995.000000	425.000000	30.000000	NaN	

8 rows × 37 columns

Now we should check to see if there is any missing value in the dataset.

```
1 total = train_value.isnull().sum().sort_values(ascending=False)
2 sum(total)
```

```
0
```

The result is 0, which is a nice dataset collected. So we do not have to worry about handling missing data.

As begin, I scrutinized the description of each variable provided on the website to try understand the dataset by my intuition. But I quickly realized that I was still not very sure which variable I should add into my training model. Therefore, I used a shortcut to reduce the dimension of the data set, which is Prinicipal Component Analysis.

Data Clean up and Preparation

Other than straightly apply PCA, since the first column is the building_id, I prefer to exclude it from my training model. Also, there are 8 objects types data in the dataset, we need convert those features into numeric if we decided to include any of those into our machine learning training model later.

I used LabelEncoder package in python to convert all the categorical variables into numeric variable. After the previous importance step, I applied the decomposition model from the sci-kit learning library. I set 'n_components' equal to 0.99999, it returns me 4 features.

I then transformed the fit of PCA and separated datasets with 70 percent for training and 30 percent for testing, splited into X_train, X_test, y_train, y_test for training preparation.

Last but not least in this stage -- the standardization step. There are different ways of doing the standard scaling, for example, in the 'preprocessing' module of python, there are RobustScaler and StandardScaler. I used the StandardScaler class from the preprocessing module of the Sci-kit learn library to transform both the X_train and X_test.

Testing

For testing purpose, I chose to use Support Vector Machine to acquire a quick trained model to get result back for my feature selection since I read about article about Support Vector Machine and how it was used in self driving car area due to its fast computation feature. I will use F1 score to measure my model selection. With PCA and SVM, I get a 0.573 F1 score for the SVC model after a principal component feature deriving, which is not satisfied.

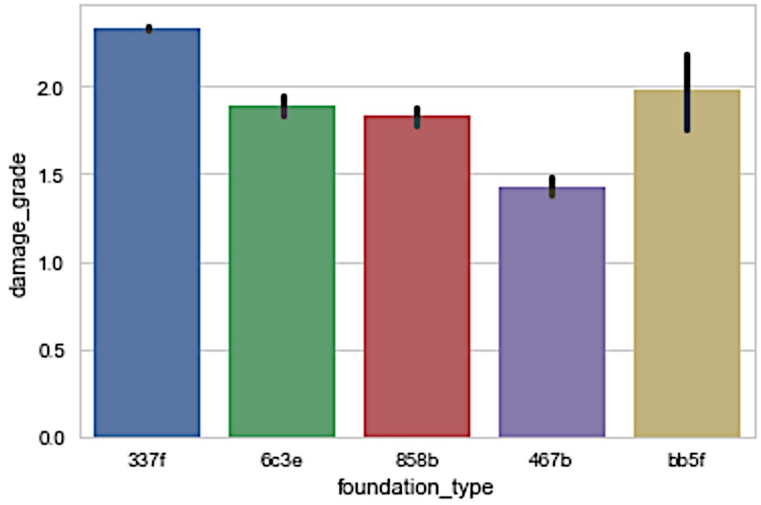
I reckon that I need to change something with my approach, either from the data preparation part or the algorithm selection part, or both.

I started with a quick testing by using Random Forest Classification machine learning algorithm with the feature I selected to check if I could get a better result than using Support Vector Machine. It returns 0.613, which is not satisfactory either.

With the quick testing of using two different machine learning model: Support Vector Machine and Random Forest Classification, I get F1 scores around 60% percent accuracy, and this signals me that I need to do a better job on the data preparation and feature selection stage.
Hence, I slowed down my pace and go back to the data preprocessing stage. But this time, instead of using quick tools like Principal Components or feature elimination algorithm, I decided to get into each categorical data and look at their relations with our target variable 'damage_grade'.

For convenience, I already merged the 'train_value' with the 'train_label' dataset, so that I can see how those categorical data related to the damage level.
By my understanding, foundation type will affect the grade of damage once earthquake hits a building. So I made a bar plot to see the relation between 'damage_grade' and 'foundation_type'.

```
<matplotlib.axes._subplots.AxesSubplot at 0x1150a08d0>
```



We can see that 337f is the most infirm foundation type dealing with earthquake, 858b is the most robust type within the five types dealing with earthquake. So I would guess for those building_id who has type 337t would more likely to get a damage_grade of 3 and type 858b would more likely to get a damage_grade of 1.

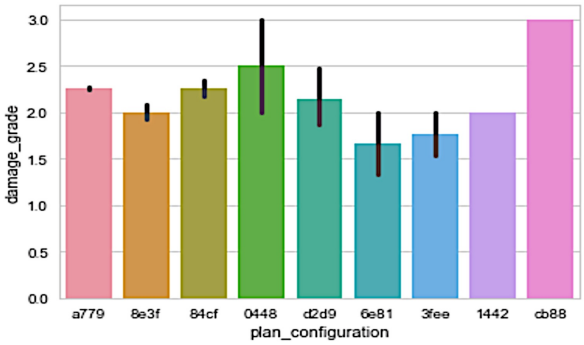
Since 'foundation_type 'matters with 'damage_grade', I would also speculate 'land_surface_condition' may also affecting the 'damage_grade', so would like to check their relation.

```
land_surface_condition  damage_grade
2f15                    1                 35
                        2                191
                        3                121
808e                    1                102
                        2                821
                        3                419
d502                    1                801
                        2               4624
                        3               2886
dtype: int64
```

From the grouping count we can see that 'd502' are the most common type among the three types of land surface condition, '808e' is the second most common, despite the ratios of the counting value with each of its damage grade among the three types looks close, I decided to include 'land_surface_condition' in my model since it makes common sense to have this feature to predict damage grade.

```
1 sns.barplot(x='plan_configuration', y='damage_grade', data=result)
```

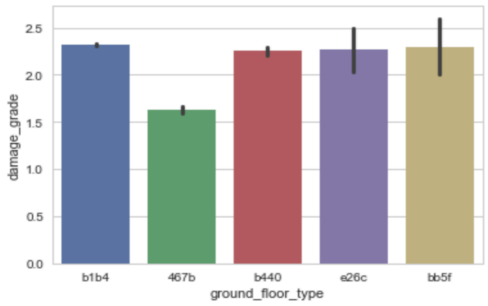
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a1e3138d0>
```



I then looked at the 'plan_configuration' , and find that it actually has impact on 'damage_level', so this will be one of the right candidates for my machine model.

```
1 sns.barplot(x='ground_floor_type', y='damage_grade', data=result)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a1ae9d090>
```

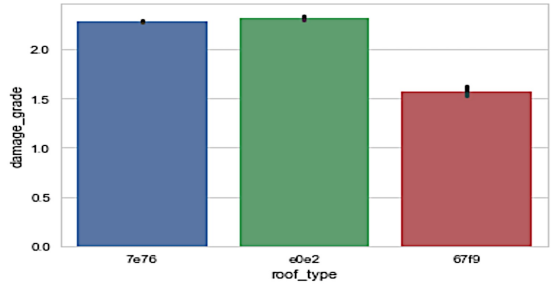


For 'ground_floor_type', since it does not show it has significant different impact on 'damage_grade', I will exclude it from the training model for now.

Then I took a look at the 'roof_type' with 'damage_grade', it looks like a building where has a roof_type e0e2 will tend to suffer a higher level of damage, whereas roof type 67f9 is more resistant towards earthquake, so I would include 'roof_type' into my training model.

```
1 sns.barplot(x='roof_type', y='damage_grade', data=result)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a1c1da510>
```



Feature Determination

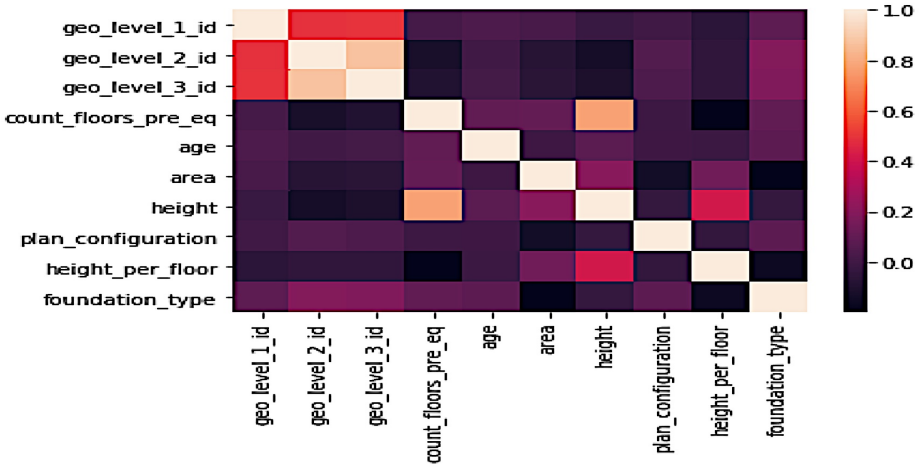
I am eager and somewhat reckless to test my assumption. So if all my assumption is correct, the tentative model should be good to go.
So far, I included 'geo_level_1_id', 'geo_level_2_id', 'geo_level_3_id', 'count_floors_pre_eq', 'age', 'area', 'height', 'land_surface_condition', 'foundation_type', 'plan_configuration', 'roof_type' into my model.

I repeated the same process as previous, instead, this time I did not use simple principal component analysis to help me select features for my model but the ones I just mentioned above. This time, with default parameters of Random Forest Classifier, I get 61.6% accuracy rate.

By looking at the bar chart I created for 'plan_configuration', we can tell that different type of plan configuration will have different impact resisting earthquake. So I assign different numeric level to each type of plan configuration.

My turned the original values of 'foundation_type' into 1,2,3,4,5 accordingly to their resistance of earthquake. The more robust the foundation type towards earthquake, the higher the number they will get. Same as 'foundation_type', I convert the character value of 'plan_configuration' into 1,2,3,4,5,6,7,8,9 in terms of their contribution to 'damage_grade' showing in the graph above.

I also generated a variable one is called 'height_per_floor' from a division of 'height' and 'count_floors_pre_eq' , looking at the correlation heat map created, we can see there is significant relation between 'height' and 'count_floors_pre_eq', so I will include 'height_per_floor' into my model.



Model Selection

With the feature I selected for model training, I test them with several different machine learning approach to predict the 'damage_grade', they are Naïve Bayes, KNN, Support Vector Machine, Random Forest Classifer, XGboost. With default parameters on all the models in sci-kit learn, it turns out that Random Forest classifier and gradient boosting classifier return similar result, where the two can out perform KNN or Support Vector machine by approximately up to 10% accuracy. Below shows the model score comparisons with default hyper-parameters in sci-kit learn and XGboost libraries.

Naïve Bayes: 0.5645
Support Vector Machine: 0.5969
Random Forest Classifier: 0.6575
KNN: 0.5825
Gradient Boosting: 0.6623
XGboost Classifier: 0.6742

Since the XGboost Classifier achieved the highest score, I focus the hyper parameter tuning with this model. To try to maximize the performance of the XG boost model, I set both the hyper parameter 'colsample_bylevel' and 'colsample_bytree' to 0.7, learning rate to 0.05, number of estimator to 2000, subsample to 0.5. With this setting, the model returns me a 0.6774 prediction score.

Conclusion

Accuracy of prediction largely varies depends on our data clean up and data preparation processing. Specifically, during the data preparation stage, we should explore the data and do some data munging, data wrangling work. In the Nepal damage project, we dropped the 'building_id' for model training purpose, converted categorical variables into numeric, replaced varchar values in 'plan_configuration' in into integer, added new variable that

generated from height and floor counts. After we spited data into 70 percent for training and 30 percent for testing, we standardized out training and testing data, then put them into different machine-learning models and compared results. In this project, we can see that tree and boosting algorithms like XGboost Classifier and Random Forest can be very effective for classification analysis with the correct data feature selected.