Richter's Predictor: Modeling Earthquake Damage

Chukwudera Mojekwu
Ferdous Alam
Kevin Lu
Steve Hall

Background

- In April 2015, a 7.8 magnitude earthquake devastated Nepal
- Nearly 9,000 lives were lost
- Millions of people were left homeless
- Damages totaled about half of Nepal’s GDP! (~$10B USD)

Research Motivation

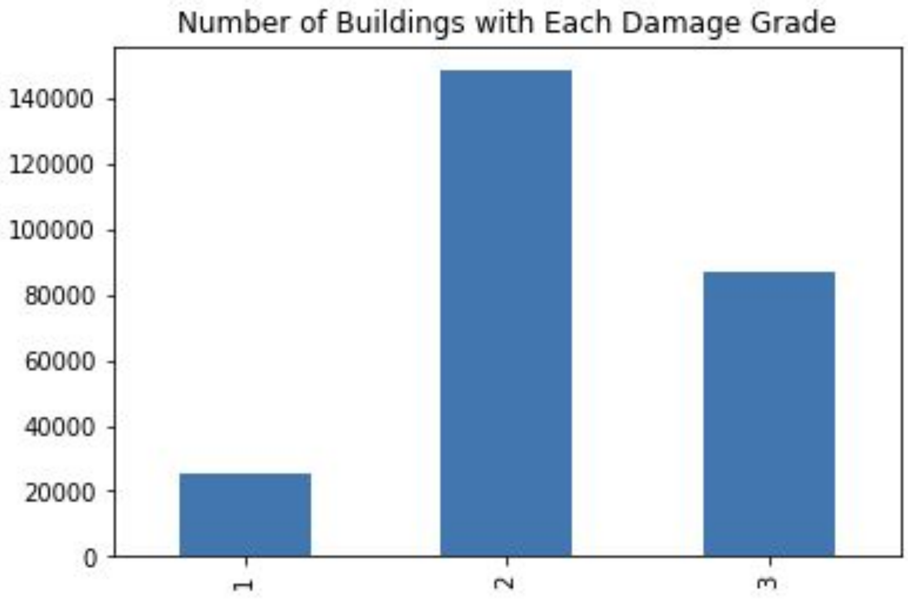
- Since the earthquake, the Nepalese National Planning Commission has **created the largest post-disaster dataset ever** to help with the planning and rebuilding of the affected districts

- Our **goal is to predict the level of damage a building suffered** as a result of the 2015 earthquake using aspects of the buildings location and construction

- The model developed can be used by the Nepal Government's to allocate resources disaster relief and rebuilding program

Data - Response variable

- **Building Damage Grade**
- Categorical based on the damage of each building
- Level 1 - 25,124 (9.6%)
- Level 2 - 148,259 (56.9%)
- Level 3 - 87,218 (33.5%)



Data - Predictor variables

Shape of training data frame: (260601, 38)

geo_level_1_id	int64
geo_level_2_id	int64
geo_level_3_id	int64
count_floors_pre_eq	int64
age	int64
area_percentage	int64
height_percentage	int64
land_surface_condition	object
foundation_type	object
roof_type	object
ground_floor_type	object
other_floor_type	object
position	object
plan_configuration	object
has_superstructure_adobe_mud	int64
has_superstructure_mud_mortar_stone	int64
has_superstructure_stone_flag	int64
has_superstructure_cement_mortar_stone	int64
has_superstructure_mud_mortar_brick	int64

has_superstructure_cement_mortar_brick	int64
has_superstructure_timber	int64
has_superstructure_bamboo	int64
has_superstructure_rc_non_engineered	int64
has_superstructure_rc_engineered	int64
has_superstructure_other	int64
legal_ownership_status	object
count_families	int64
has_secondary_use	int64
has_secondary_use_agriculture	int64
has_secondary_use_hotel	int64
has_secondary_use_rental	int64
has_secondary_use_institution	int64
has_secondary_use_school	int64
has_secondary_use_industry	int64
has_secondary_use_health_post	int64
has_secondary_use_gov_office	int64
has_secondary_use_use_police	int64
has_secondary_use_other	int64

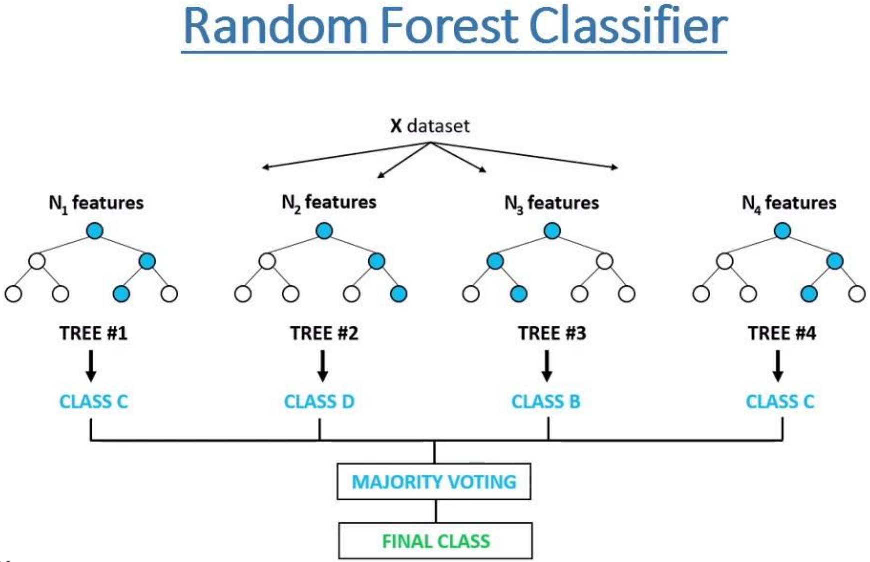
Feature Engineering

- StandardScaler
- Log Transformations
- Label Encoding
- One Hot Encoding
- High Multicollinearity
- Adjusted for outliers
- Practical Significance

Feature Engineering cont.

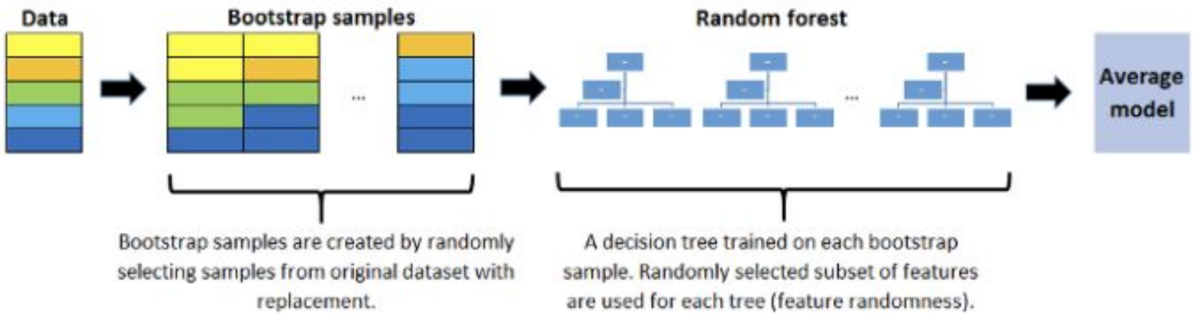
- Removed features
  - High correlation variables - 'has_superstructure_mud_mortar_stone', 'count_floors_pre_eq’
  - Non relevant variables - 'count_families', 'legal_ownership_status_a', 'legal_ownership_status_r', 'legal_ownership_status_v', 'legal_ownership_status_w'
- Encoding
  - One Hot Encoding - 'land_surface_condition', 'foundation_type', 'roof_type', 'ground_floor_type', 'other_floor_type', 'position', 'plan_configuration'
- Statistical
  - Removed Outliers from the top and bottom 5% based on ‘age’ variable

Baseline Model



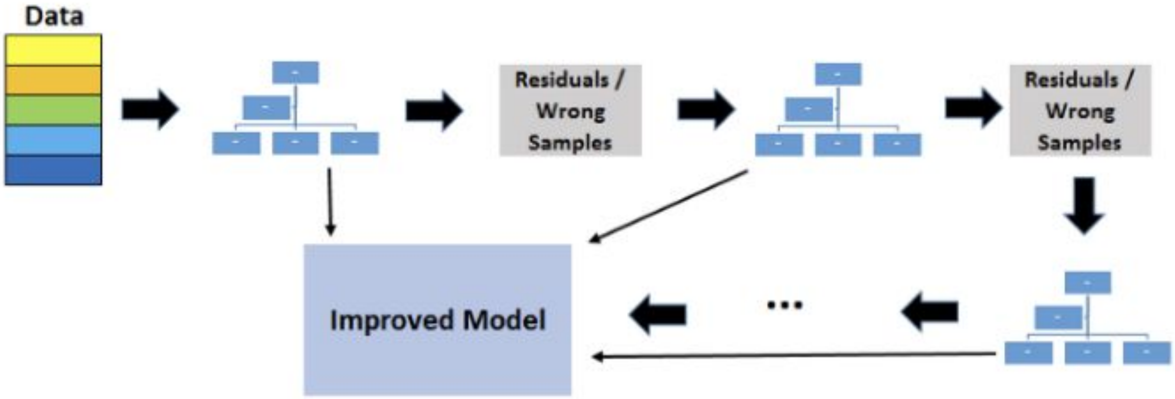
- StandardScaler
- RandomForestClassifier
- Used One Hot Encoding for data type objects
- Baseline Accuracy of 58.15%

Bagging



Random Forests

Boosting



Gradient Boosted Decision Trees

Source: https://towardsdatascience.com/gradient-boosted-decision-trees-explained-9259bd8205af

Algorithm selection

- We performed a high-level grid search across a number of algorithms
- Boosting (XGBoost and LGBM) > bagging (RandomForest and ExtraTrees)
- XGBoost outperformed LGBM modestly in all grid searches

Model	Accuracy	Balanced Accuracy	F1 Score
XGBoost	0.72524	0.6267	0.716478
LGBM	0.706143	0.597645	0.694756
RandomForest	0.601016	0.405312	0.496077
LogisticRegression	0.578401	0.423155	0.484802
ExtraTrees	0.574372	0.362573	0.429528

*Note this table only shows results of one of several initial grid searches across algorithms

XGBoost hyperparameter tuning

Name	Description	Default	Best	Comments
max_depth	Maximum depth of each tree (same as DecisionTreeClassifier)	6	10	12 performed better in our test but worse in the contest (likely due to over-fitting)
min_child_weight	Minimum sum of instance weight needed in a child (i.e. minimum number of instances in a node)	1	5	Also used to prevent over-fitting (e.g. higher values prevent model from learning very specific relationships)
eta	Learning rate (lower = slower, but requires more trees to find the “optimal” solution)	0.3	0.3	A lower learning rate allows the model to become more robust and generalized

Final results

- Our best model resulted in a F1 score of 0.7436 (Top model 0.7558)
- Ranked in the top 8%! Not too shabby
- What else would you suggest?
- How would you approach this problem?

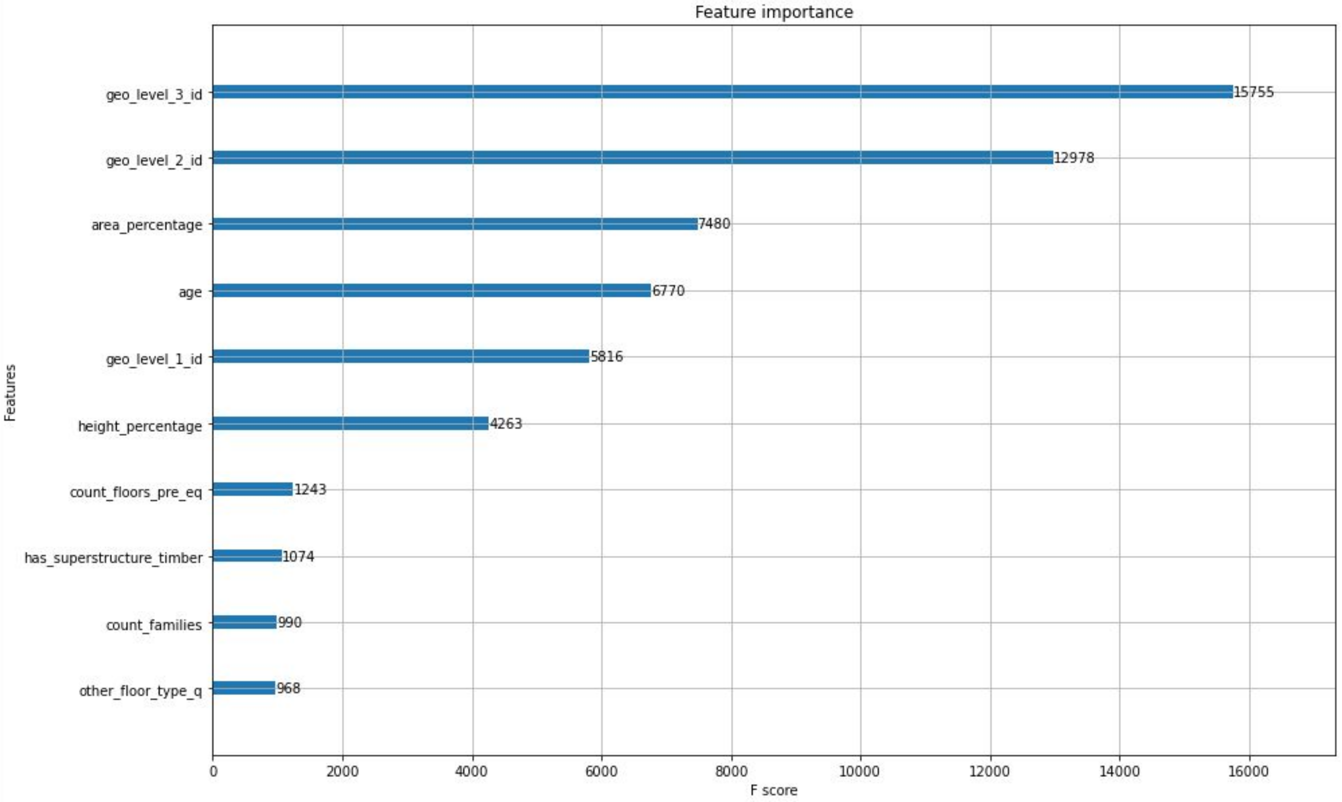
Conclusions and next steps

- Feature engineering led to larger information gains than parameter tuning
  - Feature engineering boosted the f1 score by ~9%
  - Hyperparameter tuning increased the f1 score by ~3%
- Tree algorithms outperformed in general
- Boosting methods outperformed bagging classifiers

For next steps,

- Testing additional algorithms
  - Deep learning
  - Stacking classifiers
- Consult with subject matter experts
  - Feature importance
  - Additional factors

Appendix:



Feature importance
geo_level_3_id
geo_level_2_id
area_percentage
age
geo_level_1_id
height_percentage
count_floors_pre_eq
has_superstructure_timber
count_families
other_floor_type_q
15755
12978
7480
6770
5816
4263
1243
1074
990
968
Features
0
2000
4000
6000
8000
10000
12000
14000
16000
F score