**Housing Damage Estimator for Earthquakes: An Automated Machine Learning Application for 2015, Nepal Earthquake**

Emre Ozmen, PhD, PMP

1. **Gorkha (2015, Nepal) Earthquake**

Gorkha earthquake (also known as the Nepal earthquake) with a magnitude of 7.8, nearly 10,000 people died and 20,000 injured on the 25th of April, 2015. It is reported as the worst natural disaster after the 1934 earthquake. Due to its center and occurrence (11:56am) time, as a blessing, it is recorded that both the capital of Nepal and outdoor rural workers were relatively less effected than it normally can. [1] However, it did not prevent hundreds of thousands of Nepalese from being homeless for months and outbreaks. It also triggered activities on Mouth Everest and avalanches were followed by. [2] The effects on rebuilding economy were calculated around $5billion dollars.

1. **Post-disaster Data Collection and Challenge**

To scrutinize the facts about its damage on particular of housing, National Planning Commission Secretariat of Nepal worked with Kathmandu Living Labs and the Central Bureau of Statistics and conducted a survey. It ended with one of the largest post-disaster data collection ever attempted, consisting of valuable information on earthquake statistics, household conditions, and socio-economic-demographic impact. Commission declared that the data collected using mobile technology first targeted to identify beneficiaries eligible for government help, but it also served researcher, other governments.

On year 2020, DrivenData Inc. that is known with hosting data science challenges for intermediate-level practices, started a competition about 2015 Nepal earthquake’s household damage modelling. [3] The dataset consists of around 300K rows (houses) with 39 labels and mostly refers to building characteristics. Data type is dominated by binary, where 30% refers to int and categorical. The target of data set as damage\_grade and it is represented out of 3 per ordinal order, where 3 tied with serious

The remaining 38 labels are described as below:

* categorical: There are 7 columns in this data type
  + 3 of them are linked with foundational attributes, gradually differs from 3-to-6 categories
  + 3 of them are linked with floor/roof types that were used, gradually differs from 3-to-4 categories
  + 1 refers to ownership status with 4 categories
* int: There are 8 columns in this data type
  + 3 of them are linked with geospatial location, gradually from 1-to-3 refers to density
  + 4 of them are attributes with dimensions of the house, in respect to height, area as well as # of floors
  + 1 refers to household numbers
* binary: There are 25 columns in this data type
  + 7 of them are linked with engineering level
  + 7 of them are linked with material types; stone, mud and its derivatives, mortar and its derivatives, timber and its derivatives
  + 7 of them are linked with functional types; government, hospital, education, resorts etc

1. **Performance Metrics**

The fact that the target is based on an ordinal number, both categorization and regressions techniques can be applied. Although performance metrics may depend on model chosen, low error, high precision/recall approach will be applied as a holistic approach. F1 as a demonstration of both precision/recall can be interpreted as one the scores. F1 is defined as:

1. **Automated Machine Learning Methodology Exploration**

Machine learning is a taxonomy created to replace the existing taxonomy with predictions. [4] However, there is one major distinction between them. The former is more event based, where the latter is more rule based. In other words, to make some predictions you either set rules or train your model with what happened without rules. [5, 6] Knowing that the polarization between regression and classification is not always needed as mentioned earlier, the possibility of discovering a huge data (like we have here, 300K houses with 40 columns) with no bindings excites practitioners. However, practicality perspective setting boundaries so wide might jeopardize study in the sake of necessity of dealing with more than 10 different models from regression and classification domains. [7] From this regard, automated machine learning (AML) concept can be utilized to both have the advantages of omni-model environment and test AML’s performance.

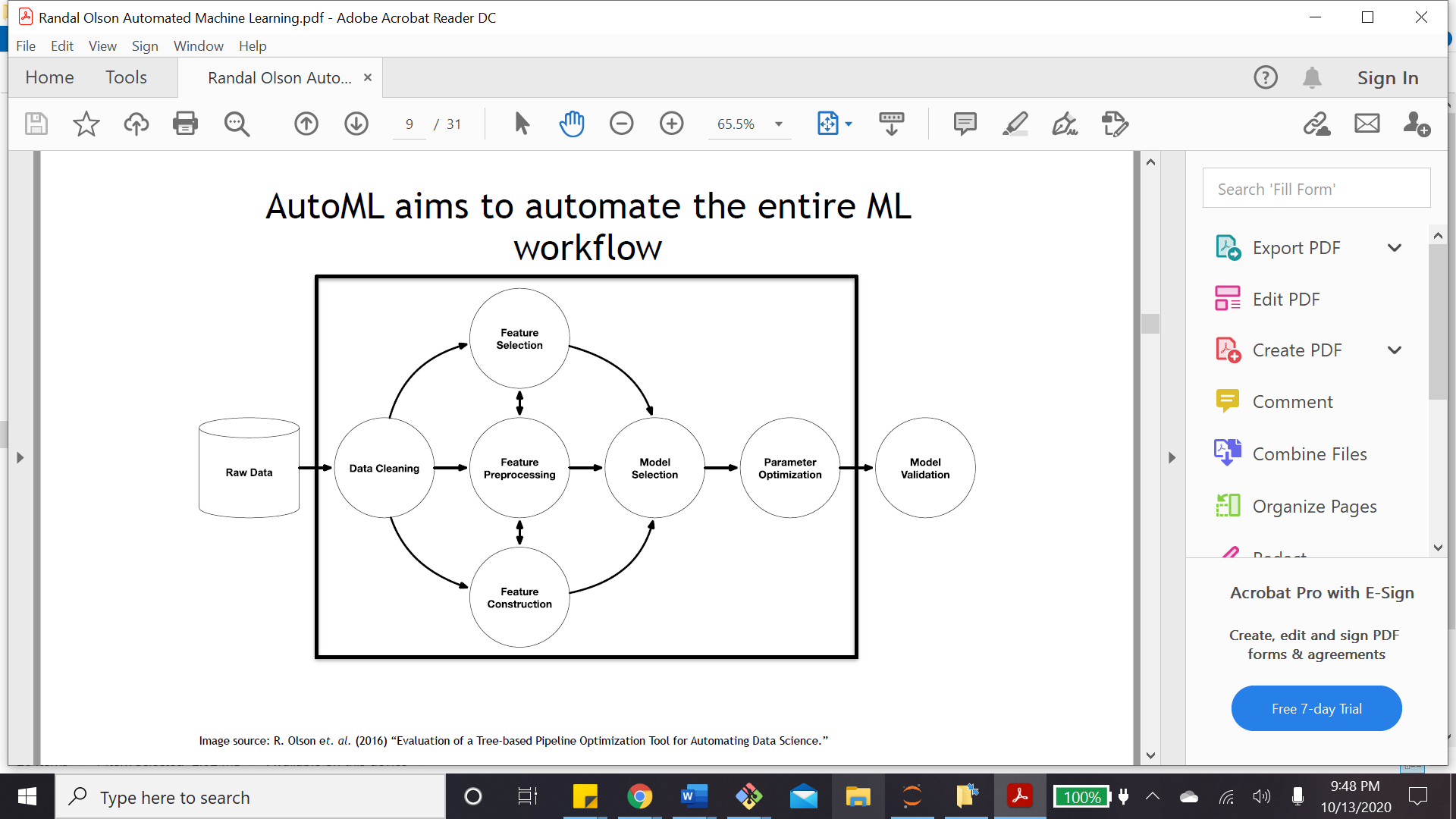


Figure 1: Tree-based pipeline (Olson, 2016)

A typical machine learning scientist’s list of tasks vary from data gathering to visualization, usually requires significant time for data cleansing, feature processing per construction and selection, model selection with more than 10 options maybe 20 with its derivatives, parameter optimizations, model validation and maybe even production. [8] Most major companies created different roles for this group of iterative tasks such as including data curator, data steward, data engineer, data architect and data scientist. If this is not the case, AML can help one individual with many hats. In the end, AML’s commitment is to manage all except raw data and model validation. To be specific:

* Works with many models including Boost, Naïve ayes, Decision Tree, Random Forest, Linear Models, Gradient Descent, Logistic Regression, Multinomials, Support Vector
* Adjusts default parameters in a way to find the best fit
* Finds best algorithms
* Optimize the entire workflow, multi-arm bandit

There are many attempts in respect to AML, in both open source and commercial arena, to mention few for the former:

* auto-Weka is a Java library, built on Weka
* auto-sklearn is a Python library, optimizes per Bayesian
* TPOT works with Python
* auto-keras is a Python library, has very powerful classification/regression models for not only structured data, but also images and texts
* H20 AutoML is developed with Java, works with Python, R and Scala

1. **Application with H20 AutoML**

Amongst all, H20 AutoML has distinctive features per three aspects, it is explicit in terms of model names (and flexible in terms of inclusions or exclusions), gives confusion matrix if it applies and proposes important factors. [9] To be more specific, H20 AutoML requires only two data and two stopping parameters. On the other hand, it handles a total of 27 parameters to burst the control on user hands. [10]

First H20 is initiated.

**import** **h2o**

**from** **h2o.automl** **import** H2OAutoML

h2o.init()

Checking whether there is an H2O instance running at http://localhost:54321 ..... not found.

Attempting to start a local H2O server...

; Java HotSpot(TM) 64-Bit Server VM (build 25.261-b12, mixed mode)

Starting server from C:\Users\emreo\miniconda3\lib\site-packages\h2o\backend\bin\h2o.jar

Ice root: C:\Users\emreo\AppData\Local\Temp\tmpnjlg22vp

JVM stdout: C:\Users\emreo\AppData\Local\Temp\tmpnjlg22vp\h2o\_emreo\_started\_from\_python.out

JVM stderr: C:\Users\emreo\AppData\Local\Temp\tmpnjlg22vp\h2o\_emreo\_started\_from\_python.err

Server is running at http://127.0.0.1:54321

Connecting to H2O server at http://127.0.0.1:54321 ... successful.

|  |  |
| --- | --- |
| H2O\_cluster\_uptime: | 06 secs |
| H2O\_cluster\_timezone: | America/New\_York |
| H2O\_data\_parsing\_timezone: | UTC |
| H2O\_cluster\_version: | 3.30.0.6 |
| H2O\_cluster\_version\_age: | 3 months and 3 days |
| H2O\_cluster\_name: | H2O\_from\_python\_emreo\_oxt8zm |
| H2O\_cluster\_total\_nodes: | 1 |
| H2O\_cluster\_free\_memory: | 3.521 Gb |
| H2O\_cluster\_total\_cores: | 8 |
| H2O\_cluster\_allowed\_cores: | 8 |
| H2O\_cluster\_status: | accepting new members, healthy |
| H2O\_connection\_url: | http://127.0.0.1:54321 |
| H2O\_connection\_proxy: | {"http": null, "https": null} |
| H2O\_internal\_security: | False |
| H2O\_API\_Extensions: | Amazon S3, Algos, AutoML, Core V3, TargetEncoder, Core V4 |
| Python\_version: | 3.7.7 final |

Data can be automatically split into training and test dataset, however here, data already came in a split way.

In [1]:

train = h2o.import\_file(r"C:\Users\YourUsername\Desktop\train\_quake.csv")

test = h2o.import\_file(r"C:\Users\YourUsername\Desktop\test\_quake.csv")

Parse progress: |█████████████████████████████████████████████████████████| 100%

Parse progress: |█████████████████████████████████████████████████████████| 100%

Features and target were identified.

In [2]:

x = train.columns

y = "damage\_grade"

x.remove(y)

Factors set for binary classification.

In [3]:

train[y] = train[y].asfactor()

test[y] = test[y].asfactor()

* 1. **Classification with Decision Tree**

It has two options, max\_models (even if 20 was selected, 1 hour is the maximum runtime by default) or max\_runtime\_secs. First, max\_runtime\_secs was selected, since only one model was applied, Decision Tree (DRF). There are two reasons for that, first DRF has extensive reports per confusion matrix and important factors, second one model gets fast results and a quick exploration opportunity.

In [4]:

aml = H2OAutoML(max\_runtime\_secs = 60, include\_algos = ["DRF"])

aml.train(x=x, y=y, training\_frame=train)

AutoML progress: |████████████████████████████████████████████████████████| 100%

Due to single model approach, there is only one model in the leaderboard. Remembering that the campion F1 score is only 0.76, this cannot be very misleading, especially after a series of fine tuning or with regression applications.

In [5]:

*# View the AutoML Leaderboard*

lb = aml.leaderboard

lb.head(rows=lb.nrows)

| **model\_id** | **mean\_per\_class\_error** | **logloss** | **rmse** | **mse** |
| --- | --- | --- | --- | --- |
| DRF\_1\_AutoML\_20201003\_232545 | 0.414617 | 1.14618 | 0.480915 | 0.231279 |

Out[5]:

In [6]:

aml.leader

Model Details

=============

H2ORandomForestEstimator : Distributed Random Forest

Model Key: DRF\_1\_AutoML\_20201003\_232545

Model Summary:

|  |  | **number\_of\_trees** | **number\_of\_internal\_trees** | **model\_size\_in\_bytes** | **min\_depth** | **max\_depth** | **mean\_depth** | **min\_leaves** | **max\_leaves** | **mean\_leaves** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** |  | 2.0 | 6.0 | 849631.0 | 20.0 | 20.0 | 20.0 | 8552.0 | 14272 | 11251.8 |

ModelMetricsMultinomial: drf

\*\* Reported on train data. \*\*

MSE: 0.2616145729335327

RMSE: 0.511482720073252

LogLoss: 3.1327581170891725

Mean Per-Class Error: 0.43795864384785194

Confusion Matrix: Row labels: Actual class; Column labels: Predicted class

|  | **1** | **2** | **3** | **Error** | **Rate** |
| --- | --- | --- | --- | --- | --- |
| **0** | 5831.0 | 8015.0 | 1248.0 | 0.613688 | 9,263 / 15,094 |
| **1** | 4340.0 | 70191.0 | 15184.0 | 0.217622 | 19,524 / 89,715 |
| **2** | 668.0 | 24631.0 | 27127.0 | 0.482566 | 25,299 / 52,426 |
| **3** | 10839.0 | 102837.0 | 43559.0 | 0.343982 | 54,086 / 157,235 |

Top-3 Hit Ratios:

|  | **k** | **hit\_ratio** |
| --- | --- | --- |
| **0** | 1 | 0.656018 |
| **1** | 2 | 0.933539 |
| **2** | 3 | 1.000000 |

Up above is the first branch, where below shows the second branch. Accuracy, aka F1 is almost (from 0.66 to) 0.7 after 5 tours of cross-validations, which is closer to winning score.

MSE: 0.23127907948492804

RMSE: 0.480914836000022

LogLoss: 1.146178551822972

Confusion Matrix: Row labels: Actual class; Column labels: Predicted class

|  | **1** | **2** | **3** | **Error** | **Rate** |
| --- | --- | --- | --- | --- | --- |
| **0** | 10226.0 | 14145.0 | 753.0 | 0.592979 | 14,898 / 25,124 |
| **1** | 5747.0 | 124045.0 | 18467.0 | 0.163322 | 24,214 / 148,259 |
| **2** | 708.0 | 41815.0 | 44695.0 | 0.487548 | 42,523 / 87,218 |
| **3** | 16681.0 | 180005.0 | 63915.0 | 0.313257 | 81,635 / 260,601 |

Top-3 Hit Ratios:

|  | **k** | **hit\_ratio** |
| --- | --- | --- |
| **0** | 1 | 0.686743 |
| **1** | 2 | 0.957026 |
| **2** | 3 | 1.000000 |

Cross-Validation Metrics Summary:

|  |  | **mean** | **Sd** | **cv\_1\_vali** | **cv\_2\_vali** | **cv\_3\_vali** | **cv\_4\_vali** | **cv\_5\_vali** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | accuracy | 0.686743 | 0.00498712 | 0.677864 | 0.689428 | 0.688142 | 0.89121 | 0.68915 |
| **1** | err | 0.313256 | 0.00498712 | 0.322135 | 0.310571 | 0.311857 | 0.37875 | 0.31084 |
| **2** | err\_count | 16327.0 | 260.0721 | 16790.0 | 16187.0 | 16254.0 | 16203.0 | 16201.0 |
| **3** | logloss | 1.146177 | 0.207669 | 1.515895 | 1.026942 | 1.045588 | 1.02286 | 1.06017 |
| **4** | max\_per\_class\_error | 0.593018 | 0.0087694 | 0.608197 | 0.589768 | 0.585993 | 0.88855 | 0.59234 |
| **5** | mean\_per\_class\_accuracy | 0.585367 | 0.0059955 | 0.574786 | 0.587278 | 0.587304 | 0.58965 | 0.58781 |
| **6** | mean\_per\_class\_error | 0.414633 | 0.0059946 | 0.4252139 | 0.412215 | 0.412963 | 0.41045 | 0.41218 |
| **7** | mse | 0.231276 | 0.0038536 | 0.2380311 | 0.228675 | 0.229034 | 0.23067 | 0.22997 |
| **8** | r2 | 0.382017 | 0.0123747 | 0.360085 | 0.388858 | 0.387967 | 0.388377 | 0.38515 |
| **9** | rmse | 0.480016 | 0.0039866 | 0.487843 | 0.478223 | 0.478603 | 0.480168 | 0.47956 |

Scoring History:

|  |  | **timestamp** | **duration** | **number\_of\_trees** | **training\_rmse** | **training\_logloss** | **training\_classification\_error** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **0** |  | 2020-10-03 23:26:23 | 37.893 sec | 0.0 | NaN | NaN | NaN |
| **1** |  | 2020-10-03 23:26:28 | 42.262 sec | 2.0 | 0.511483 | 3.132758 | 0.343982 |

Variable importance would not significantly change form one model to another. Per top factors, geospatial location looks like dominating other. Floor type derivatives, age of the building, foundation type and area percentage are also significant attributes.

Variable Importances:

|  | **variable** | **relative\_importance** | **scaled\_importance** | **percentage** |
| --- | --- | --- | --- | --- |
| **0** | geo\_level\_1\_id | 17334.062500 | 1.000000 | 0.220961 |
| **1** | geo\_level\_2\_id | 7466.929688 | 0.430766 | 0.095183 |
| **2** | geo\_level\_3\_id | 5873.937500 | 0.338867 | 0.074876 |
| **3** | building\_id | 4998.819336 | 0.288381 | 0.063721 |
| **4** | other\_floor\_type | 4635.594238 | 0.267427 | 0.059091 |
| **5** | age | 4103.796387 | 0.236748 | 0.052312 |
| **6** | foundation\_type | 4081.254883 | 0.235447 | 0.052025 |
| **7** | area\_percentage | 3984.306885 | 0.229854 | 0.050789 |
| **8** | ground\_floor\_type | 3447.547852 | 0.198889 | 0.043947 |
| **9** | height\_percentage | 3026.006592 | 0.174570 | 0.038573 |
| **10** | count\_floors\_pre\_eq | 1828.075317 | 0.105461 | 0.023303 |
| **11** | has\_superstructure\_cement\_mortar\_brick | 1784.335083 | 0.102938 | 0.022745 |
| **12** | position | 1531.578491 | 0.088357 | 0.019523 |
| **13** | roof\_type | 1530.040283 | 0.088268 | 0.019504 |
| **14** | has\_superstructure\_mud\_mortar\_stone | 1443.995361 | 0.083304 | 0.018407 |
| **15** | count\_families | 1268.828247 | 0.073199 | 0.016174 |
| **16** | has\_superstructure\_timber | 1258.311279 | 0.072592 | 0.016040 |
| **17** | land\_surface\_condition | 1230.474854 | 0.070986 | 0.015685 |
| **18** | plan\_configuration | 966.522949 | 0.055759 | 0.012320 |
| **19** | has\_superstructure\_bamboo | 790.166809 | 0.045585 | 0.010072 |

See the whole table with table.as\_data\_frame()

Out[6]:

Due to decision tree nature, we are producing our prediction based on probabilities, where the highest determines the damage types. Here is shown a demonstration with head of the data.

In [7]:

preds = aml.leader.predict(test)

preds

drf prediction progress: |████████████████████████████████████████████████| 100%

| **predict** | **p1** | **p2** | **p3** |
| --- | --- | --- | --- |
| 3 | 0 | 0.250845 | 0.749155 |
| 2 | 0.0039452 | 0.727689 | 0.268366 |
| 2 | 0.0245312 | 0.847322 | 0.128147 |
| 1 | 0.666667 | 0.333333 | 0 |
| 3 | 0.00146907 | 0.17418 | 0.824351 |
| 2 | 0.141407 | 0.581733 | 0.27686 |
| 1 | 0.576923 | 0.423077 | 0 |
| 3 | 0.0212464 | 0.315511 | 0.663242 |
| 2 | 0.0180474 | 0.565369 | 0.416583 |
| 2 | 0 | 0.921306 | 0.0786937 |

* 1. **All Models (Classification and Regression)**

Replication for larger set of models.

In [8]:

train = h2o.import\_file(r"C:\Users\YourUsername\Desktop\train\_quake.csv")

test = h2o.import\_file(r"C:\Users\YourUsername\Desktop\test\_quake.csv")

Parse progress: |█████████████████████████████████████████████████████████| 100%

Parse progress: |█████████████████████████████████████████████████████████| 100%

In [9]:

*# left intentionally blank*

x = train.columns

y = "damage\_grade"

x.remove(y)

This time max\_models is selected with 20 models, if applies.

In [10]:

aml = H2OAutoML(max\_models=20, seed=1)

aml.train(x=x, y=y, training\_frame=train)

AutoML progress: |█

22:19:05.835: AutoML: XGBoost is not available; skipping it.

███████████████████████████Failed polling AutoML progress log: [WinError 32] The process cannot access the file because it is being used by another process: 'C:\\Users\\emreo\\AppData\\Local\\Temp\\tmpwve5ujam.csv'

█████████████████████████Failed polling AutoML progress log: [WinError 32] The process cannot access the file because it is being used by another process: 'C:\\Users\\emreo\\AppData\\Local\\Temp\\tmp9gfk0t7e.csv'

█Failed polling AutoML progress log: [WinError 32] The process cannot access the file because it is being used by another process: 'C:\\Users\\emreo\\AppData\\Local\\Temp\\tmpkukgaxmc.csv'

█Failed polling AutoML progress log: [WinError 32] The process cannot access the file because it is being used by another process: 'C:\\Users\\emreo\\AppData\\Local\\Temp\\tmppmq\_so3d.csv'

█| 100%

* 1. **Winning Model**

Leaderboard.head brings us the top ten, Stacked Ensemble with regression leads the run, our DRF makes only a number 6 here.

In [11]:

aml.leaderboard.head()

| **model\_id** | **mean\_residual\_deviance** | **rmse** | **mse** | **mae** | **rmsle** |
| --- | --- | --- | --- | --- | --- |
| StackedEnsemble\_AllModels\_AutoML\_20200729\_22 | 0.200279 | 0.44752 | 0.20027 | 0.3507 | 0.14587 |
| StackedEnsemble\_BestOfFamily\_AutoML\_20200729 | 0.200869 | 0.44818 | 0.20086 | 0.3517 | 0.14610 |
| GBM\_grid\_\_1\_AutoML\_20200729\_221905\_model\_4 | 0.201596 | 0.44899 | 0.20159 | 0.3550 | 0.14655 |
| GBM\_5\_AutoML\_20200729\_221905 | 0.206886 | 0.45484 | 0.20688 | 0.3612 | 0.14858 |
| GBM\_grid\_\_1\_AutoML\_20200729\_221905\_model\_2 | 0.208152 | 0.45623 | 0.20815 | 0.3593 | 0.14891 |
| DRF\_1\_AutoML\_20200729\_221905 | 0.209826 | 0.45806 | 0.20982 | 0.3649 | 0.14954 |
| GBM\_4\_AutoML\_20200729\_221905 | 0.210362 | 0.45865 | 0.21036 | 0.3672 | 0.14962 |
| GBM\_3\_AutoML\_20200729\_221905 | 0.214197 | 0.46281 | 0.21419 | 0.3724 | 0.15103 |
| XRT\_1\_AutoML\_20200729\_221905 | 0.214246 | 0.46286 | 0.21424 | 0.3760 | 0.15080 |
| GBM\_2\_AutoML\_20200729\_221905 | 0.217049 | 0.465885 | 0.217049 | 0.37613 | 0.152045 |

Out[11]:

Unlike probabilities of DRF, prediction makes float numbers here. To be able to submit, they were rounded to up.

In [12]:

pred = aml.predict(test)

pred

stackedensemble prediction progress: |████████████████████████████████████| 100%

| **predict** |
| --- |
| 2.80101 |
| 2.23104 |
| 2.16015 |
| 1.26562 |
| 2.80709 |
| 2.07994 |
| 1.33965 |
| 2.91986 |
| 2.16437 |
| 2.10983 |

Out[12]:

Predictions were exported, results for F1 performance was published as 0.74, a Top 10% score out of 3,300 participants.

In [13]:

h2o.export\_file(pred, 'C:**\\**Users**\\**emreo**\\**Desktop**\\**equake2\_competition.csv')

Export File progress: |███████████████████████████████████████████████████| 100%

In [14]:

perf = aml.leader.model\_performance(test)

perf

ModelMetricsRegressionGLM: stackedensemble

\*\* Reported on test data. \*\*

MSE: 0.2040952424641873

RMSE: 0.4492346017670735

Precision: 0.773808480433232

Recall: 0.71268232018074937

F1: 0.73755288647432567

Out[14]:

1. **Conclusion**

Earthquakes are moments that make us powerless and speechless. This is even more true for emerging countries, since most of the time resources are limited. From this perspective any contribution may have say. This automated machine learning application needs to be considered in this extent. AML performed significantly well, scored Top 10% amongst 3,300 participants on 2015 Nepal Earthquake Damage Estimator with 300K houses, with very modest data cleansing and/or coding necessity. Tool was explicit enough to share important factors and very promising confusion matrix as well as cross validation derivatives. It looks like it is dependable and reproduceable to serve both practitioners and researchers. Future directions include more applications with all other countries suffering from earthquakes including Turkey, China, Iran, Chile and Indonesia.