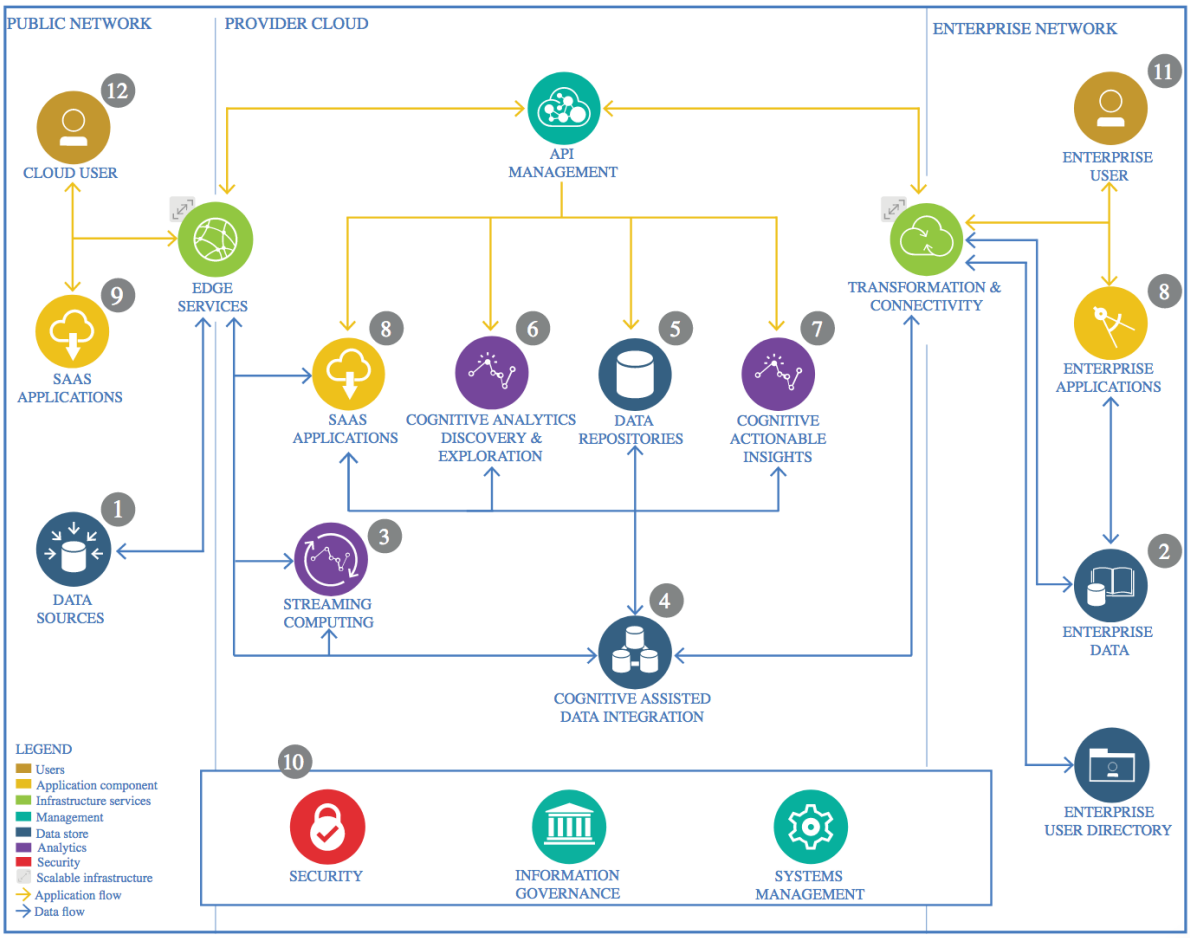
# Shaking Earth: LANL Earthquake Prediction Challenge

## Architectural Decisions Document (ADD)

Nelli Fedorova

https://www.linkedin.com/in/nelli-fedorova-7710b01a/

## 1 Architectural Components Overview




IBM Data and Analytics Reference Architecture. Source: IBM Corporation

## 1.1 Data Source

### 1.1.1 Technology Choice

In LANL Earthquake Prediction Challenge https://www.kaggle.com/c/LANL-Earthquake-Prediction/overview we predict time left to the next laboratory earthquake based on seismic signal data.

Training data is one signal of length ~629,100,000 (~150,000 x 4194), with 17 earthquakes. In test data we have 2624 separate chunks of length 150,000, for each of which we need to predict time to failure. Test result performance measure is MAE (mean absolute error).

The problem seems to be really hard to solve. As of August 2019, the ground truth of the testing dataset was not made public yet. The best public score evaluated on an unknown subset of test samples was MAE =1.08022. This score should be compared with the mean time of ~5.68 and standard deviation of ~3.67 of the original training data. If training samples were correctly classified into just 3 classes, the MAE would be around 1.16. It is realistic to expect to be able to classify test samples into ~2...3…4 classes at most.

Test samples do not form a sequence, so this problem can be approached as regression, classification, clusterization and data compression, and not as a time series prediction problem.

First, I am going to compress and transform 150,000 original inputs using signal processing and signal compression techniques like FFT and DWT up to $17^{th}$ level.

Next, I will compute 100…1000 statistics and norms for the original signal, its FFT coefficients and DWT A and D vectors.

Next, I will choose 10…100 most informative descriptors to be used in data modeling.

Then I will perform the following iterative steps.

- Step 1. Use a subset of training data to buid KNN regression model and predict TIME for TEST data points.
- Step 2. Use use nearest neighbors found by KNN to buid FF, XGB, RCV, SCR, LGB models with different model parameters (learning rate, depth, number of hidden neurons, etc.) and predict TIME for TEST points using these models.
- Step 3. Choose a new subset of training points in the vicinity of the nearest neighbors and repeat modeling from step 1.

I will use the following software tools and libraries.

- Python pandas, numpy, matplotlib, scikit-learn, etc.
- Python libraries for DWT, FFT, MFCC
- MATLAB scripts and Python Jupyter Notebooks on my laptop for preliminary data analysis, visualization and model training using small subsets of the original data

- IBM Watson Studio and Python libraries like keras and SystemML for data preparation and training final models using big data

### 1.1.2 Justification

Preliminary data analysis can be done on my laptop. I might need IBM Watson Studio with Spark, keras, etc. at later stages for training and cross-validating models on larger datasets.

## 1.2 Enterprise Data

### 1.2.1 Technology Choice

I will store all data on my laptop and back up it dailly to the external harddrive. I will use IBM cloud temporary storage while training final models.

### 1.2.2 Justification

All data can fit into my computer.

## 1.3 Streaming analytics

### 1.3.1 Technology Choice

### 1.3.2 Justification

## 1.4 Data Integration

### 1.4.1 Technology Choice

The data is clean. I just need to convert it to MATLAB binary format to speed up loading/reading in Python.

I will split the original train.cvs file of size 8.89 GB (9,555,558,244 bytes) into 4194 subsequent fragments and save them as MATLAB binary files /Train/train*.mat.

I will unzip test.zip and save 2624 csv test files as MATLAB binary files /Test/seg*.mat.

### 1.4.2 Justification

Compared to cvs, loading/reading binary MATLAB files to Python or MATLAB is much faster. Data in MATLAB format can be easily processed both in Python and MATLAB.

## 1.5 Data Repository

### 1.5.1 Technology Choice

#### *1.5.1.1 Original data storage*

- /Train with 4194 MAT files train<segment ID>.mat with 4194 subsequent fragments of training data.
- /Test with 2624 MAT files <seg_<segment ID>.mat with 2624 test fragments.

#### *1.5.1.2 Modeling results storage*

Each model will have a separate working directory with the following files.

- Python and MATLAB scripts
  - shaking_earth_etl.ipynb
  - shaking_earth_clusters.ipynb
  - shaking_earth_model.ipynb
- Data files for building models
  - X_tr.csv or X_tr.parquet – input features for training samples
  - y_tr.csv or y_tr.parquet – time to earthquake for training samples
  - X_tr_folds.csv – first and last indeces for 17 folds (earthquakes) in X_tr and y_tr
  - X_TEST.csv - input features for 2624 test samples
  - X_te.csv – inputs for 4194 samples used for internal testing
  - y_te.csv – time to earthquake for 4194 samples used for internal testing
  - X_te_folds.csv – first and last indeces for 17 folds (earthquakes) in X_te and y_te
- Modeling results
  - z_TEST.csv – predictions for 2624 test samples (averaged over 17 CV models), in submission-ready format
  - z_tr.csv – 17-fold CV predictions for training samples
  - z_te.csv – prediction for 4194 samples used for internal testing (averaged over 17 CV models)
  - shaking_earth_model*.bin –models saved using joblib.dump (optional)
  - e_te.csv – 17-fold CV and test MAE summary

#### *1.5.1.3 Documentation*

- ShakingEarth_Lightweight_ADD.docx – this document
- ShakingEarth.pttx – presentation with modeling results
- ShakingEarth.docx – report with modeling results
- ShakingEarth.mp4 – video with modeling results

### 1.5.2 Justification

Original data should be stored separately from intermediate files and final data files used for building models. For repeatability and reproducibility of modeling results I need to store Python notebooks with model parameters and Python and MATLAB scripts for generating training data.

## 1.6 Discovery and Exploration

### 1.6.1 Technology Choice

It is important to have a big picture view of the data. I will use matplotlib.pyplot and other Python libraries and MATLAB functions to compare and visualize training and test data distributions and interdependecies between variables.

- Histograms and boxplots
- Scatterplots
- Scatter matrices
- Surface and bubble plots
- Correlation heatmaps

### 1.6.2 Justification

Python and MATLAB provide many tools for data exploration and visualization.

## 1.7 Actionable Insights

### 1.7.1 Technology Choice

#### *1.7.1.1 Data compression*

- FFT for the original signal of length 150,000
- DWT up to $17^{th}$ level

#### *1.7.1.2 Descriptive statistics and vector norms*

I will compute various descriptive statistics and vector norms for the original signal of length 150,000, its FFT of length 75,000, its DWT A and D at several levels, such as:

1. Mean
2. Standard deviation
3. Maximum
4. Minimum
5. Position of the maximum
6. Position of the minimum
7. Mean difference between subsequent observations
8. Mean absolute value
9. Standard deviation for the absolute values
10. Maximum absolute value
11. Minimum absolute value
12. Position of the maximum absolute value
13. Position of the minimum absolute value
14. Mean difference between absolute values of subsequent observations
15. MSE
16. L1 norm divided by L1 norm of the original signal
17. Number of peaks/outliers of different sizes

18. Histograms
19. Wavelet energy
20. Log energy
21. Shannon
22. MFCC coefficients

#### *1.7.1.3 Feature selection*

All statistics and norms computed for the original vector and its FFT, DWT A and D (see above) are likely to be highly correlated.

With highly correlated features it is okay to use an estimated regression model to predict time for test samples as long as these samples are strictly within the scope of the model, i.e. the model interpolates rather than extrapolates. In the presence of multicollinearity it is very important to ensure that training and test data overlap and have the exact same distribution. Also, with correlated predictors we can no longer make much sense of the usual interpretation of feature importance estimations provided by FF and decision tree models.

#### *1.7.1.4 Data clusterization*

Supervised learning heavily relies on the fact that training and test data follow the exact same distribution. I will use KNN and clustering techniques like K-Means and DBSCAN to find training samples that closely resemble TEST samples, e.g. are among 5 nearest neighbors of TEST samples and/or share clusters with TEST samples.

#### *1.7.1.5 Data modeling: regression*

To predict time to earthquake I will build the following regression models.

- FF
- RCV
- SCR
- LGB
- XGB

It is important to prevent overfitting by using early stopping and keeping all models as small as possible, e.g. by limiting the depth of decision trees and the number of hidden neurons for FF networks.

#### *1.7.1.6 Data modeling: multiclass classification*

I can discretize the output into 3…10 bins/categories and use them as labels, i.e. reduce regression to a classification model. I can use the following multi-class classifiers:

- FF
- XGB

### 1.7.2 Justification

Test data and training data distributions seem to be somewhat different. I need to carefully select predictors and samples used for building models. I also need to limit the size of the models, use early stopping and perform cross-validation to prevent overfitting.

## 1.8 Applications / Data Products

### 1.8.1 Technology Choice

The results of these project will be as follows.

- (Late) submission to Kaggle. Submission file should be Unix EOL (LF) cvs file.
- PPT presentation with modeling results
- Video presentation of modeling results
- Jupyter Notebooks and MATLAB scripts

### 1.8.2 Justification

This is a purely academic research project.

## 1.9 Security, Information Governance and Systems Management

### 1.9.1 Technology Choice

### 1.9.2 Justification