ALMA MATER STUDIORUM - UNIVERSITA DI BOLOGNA

SCUOLA DI INGEGNERIA E ARCHITETTURA

Dipartimento di Informatica - Scienza e Ingegneria - DISI
Corso di Laurea Magistrale in Ingegneria Informatica

PROJECT WORK

on

DATA MINING

A solution for the LANL Earthquake Prediction challenge in Python

CANDIDATES
Valentina Protti
Giuseppe Tempesta

PROFESSOR
Prof. Claudio Sartori

Abstract

This project activity report is intended to explain the approach used in solving a Data Mining competition held on the *Kaggle* platform. In particular, the competition goes under the name "*LANL Earthquake Prediction*", and the major issue that the participants are asked to solve is to predict the time remaining before laboratory earthquakes occur, given real-time seismic data. The challenge is hosted by *Los Alamos National Laboratory* and has its ultimate goal in having the possibility to scale the results to the field, to be finally able to improve real earthquakes predictions.

The work hereby presented has its roots in Los Alamos' initial work, a first model built on laboratory experimental data. With reference to the initial data, the dataset provided for the challenge contains much more a-periodic occurrences of earthquakes, making it more realistic and comparable to real world occurrences.

The report will present the reader with an in-depth analysis of the problem and the provided data, followed by a first naïve approach to better understand the nature of the problem, and finally a comparison of the performances of various techniques for modeling the specific problem.

Contents

List of Figures

Chapter 1

Understanding the problem

Due to the huge impact of their consequences, the pursuit of forecasting earthquakes is one of the most important problems in Earth science. Studies that have been made so far focus on three key points: when, where and how large the event will be.

1.1 Previous studies

But how are these prediction achieved? Los Alamos National Laboratory has conducted a study on huge sets of laboratory experimental seismic data, showing the importance of the so called "slow earthquakes", which are still less understood. In their work [3], the researchers try to spark some light on the mechanics of slow-slip phenomena and their relationship with regular earthquakes, to which they seem to be precursors, through a complete systematic experimental study.

A second study, based on the results of laboratory experiments, takes advantages of Machine Learning techniques to predict the time to the next "labquake" by listening to the acoustic signal collected by specific laboratory sensors [6]. By using ML, even small seismic precursor magnitude can be detected, overcoming the limits of classic seismograph-based predicting systems. In particular, a Random Forest approach has been developed to predict the time remaining before the next failure, by averaging the predictions of 1,000 decision trees in each time window.

From each time window, a set of approximately 100 statistical features are computed, then selected recursively by usefulness, and lastly used to

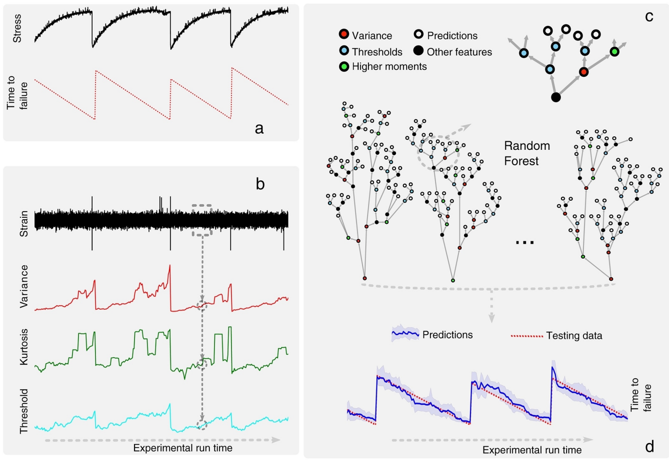

Figure 1.1: Random Forest (RF) approach for predicting time remaining before failure.

actually predict the time before the next earthquake. The results achieved through this study are quite accurate, even if it needs to be noted that a laboratory earthquake does not capture the physics of a complex, real-world earthquake.

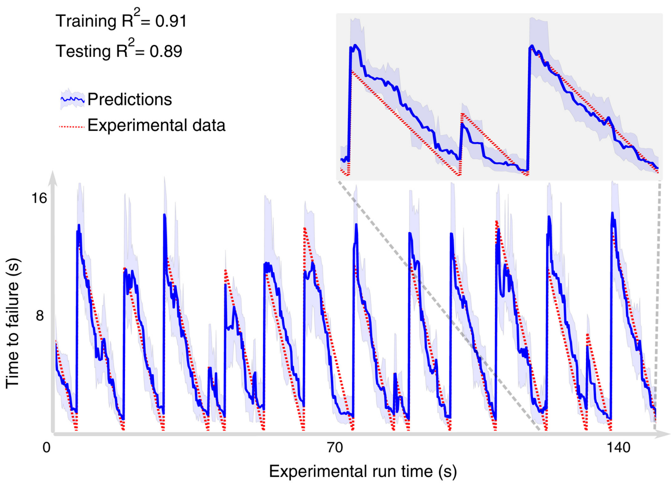

Figure 1.2: Time remaining before the next failure predicted by the Random Forest.

1.2 Understanding the data

With reference to the above mentioned studies, the dataset provided for the challenge contains more a-periodic occurrences of earthquake hazards, thus resembling more a real-world scenario. The data comes from a well-known experimental set-up used to study earthquake physics [1].

In particular, the dataset is made of two subsets:

- `train.csv` - A single, continuous training segment of experimental data (with 629.145.480 entries);
- `test` - A folder containing many small segments (`.csv`) of test data (2.624 segments of 150.000 entries each).

Each entry of the training set has two fields:

- `acoustic_data` - the seismic signal [`int16`];
- `time_to_failure` - the time (in seconds) until the next laboratory earthquake [`float64`].

On the other hand, each segment from the test set folder is named after its `seg_id` and only has one field, the `acoustic_data`. While the training set is a single, continuous, big segment of experimental data, the test set is continuous within a single segment, but the set of files cannot be considered continuous; thus, the predictions can't be assumed to follow the same pattern of the training file.

The goal of the competition is to predict a single `time_to_failure` for each segment, corresponding to the time between the last row of the segment and the next laboratory earthquake. The results must be submitted on the Kaggle platform as a `.csv` file containing the predictions for each test segment, and the score is then obtained through the application of the *Mean Absolute Error* between the real time values and the predictions.

A first approach to better understand what the data represents is to plot it (or a part of it, given the prohibitive size of the training set). In picture 1.3 we can see 1% of the training data (obtained simply by sampling every 100 entries) [5].

The following (1.4) is instead the representation of the first 1% entries of the training dataset: even at a first glance we are able to note that the

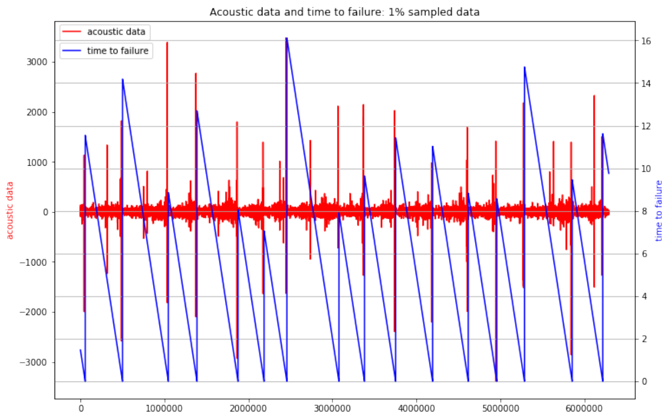

Figure 1.3: Plot of 1% sampling of the training data

failure ("*labquake*") occurs after some medium oscillations, a very large one and some other minor ones.

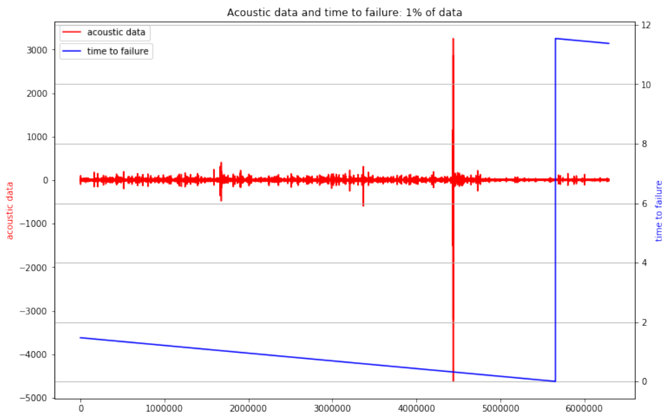

Figure 1.4: Plot of the first 1% of the training data

Before going into further details and taking a first step towards building the model from the training data, it's worth to also take a look at the structure of the test data. In picture 1.5 are represented four of the segments from the test folder [2].

Overall, what we can take away from this first dive into the datasets is:

- that the task of this Data Mining challenge will be in the regression spectrum, since the output falls in a continuous range rather than a set of discrete classes;

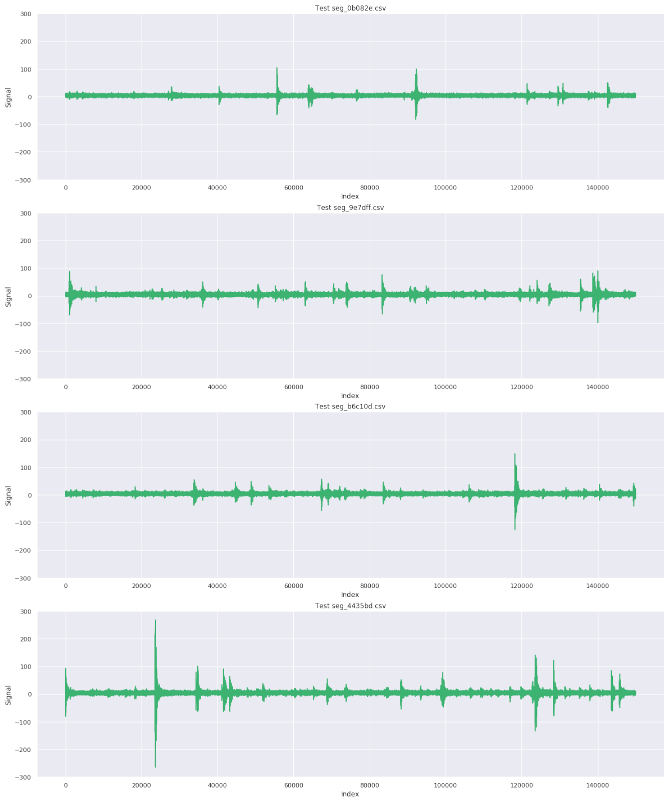

Figure 1.5: Plot of four segments of the test data

- that the dimension of the segments is not that big if compared to the very rare occurrences of laboratory earthquakes;
- that these failure events will appear very much like outliers, given the scarcity of representation and the intensity of the acoustic signal when compared to the other values.

As others participants to the challenge have noticed, it may be also relevant to note that the test set doesn't contain *any* earthquake: thus, it may be worth considering not including the few failure occurrences that can be found in the training set, to avoid the model trying to match the data to these much higher peaks when fed with the test set.

Chapter 2

A first, naïve model

A first, very simple approach to building the model for the purpose of this competition is given directly by the promoters of the challenge [4].

Before seeing what the *Python kernel* looks like, it's worth to dig a bit deeper on how the data is prepared for the task. In fact, taking the dataset "as it is", it's easy to notice that it has just one feature (`acoustic_data`) that can be used to compute the regression task of predicting `time_to_failure` on the test set.

For this reason, data needs to be prepared: the obvious choice is to divide the training set into chunks of 150.000 rows (the size of each segment of the test set; this is not the only choice available), and for each of them compute some features representing the data; in this first basic solution we will extract the mean, standard deviation, maximum and minimum. The resulting dataset will contain an entry for each portion of the initial dataset, with one column for each computed feature (in this case 4), and another dataset with just the original `time_to_failure` associated with the last row of the chunk (similarly to the test segments).

2.1 Basic Feature Benchmark

```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load in

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the "../input/" directory.
```

```
# For example, running this (by clicking run or pressing Shift+Enter) will list the files in the
      input directory

import os
print(os.listdir("../input"))

# Any results you write to the current directory are saved as output.
```

```
import matplotlib.pyplot as plt
from tqdm import tqdm
from sklearn.preprocessing import StandardScaler
from sklearn.svm import NuSVR
from sklearn.metrics import mean_absolute_error
```

```
train = pd.read_csv('../input/train.csv', dtype={'acoustic_data': np.int16, 'time_to_failure':
     np.float64})
```

After these preliminary operations of including the necessary libraries and loading the dataset, we are able to get into the data preparation as previously described. In the following snippet, `X_train` is the dataset containing the segments' 4 computed features, while `Y_train` contains the associated `time_to_failure`.

```
# Create a training file with simple derived features

rows = 150_000
segments = int(np.floor(train.shape[0] / rows))

X_train = pd.DataFrame(index=range(segments), dtype=np.float64,
                       columns=['ave', 'std', 'max', 'min'])
y_train = pd.DataFrame(index=range(segments), dtype=np.float64,
                       columns=['time_to_failure'])

for segment in tqdm(range(segments)):
    seg = train.iloc[segment*rows:segment*rows+rows]
    x = seg['acoustic_data'].values
    y = seg['time_to_failure'].values[-1]

    y_train.loc[segment, 'time_to_failure'] = y

    X_train.loc[segment, 'ave'] = x.mean()
    X_train.loc[segment, 'std'] = x.std()
    X_train.loc[segment, 'max'] = x.max()
    X_train.loc[segment, 'min'] = x.min()
```

The kernel's authors' choice for modeling the solution is to use *Support Vector Regression*. The `scikit-learn` Python library implementation of SVR recommends explicitly that the data is scaled, since Support Vector Machine algorithms are not scale invariant.

```
scaler = StandardScaler()
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)
```

```
svm = NuSVR()
svm.fit(X_train_scaled, y_train.values.flatten())
y_pred = svm.predict(X_train_scaled)
```

The results predicted are then compared with the actual training data, by computing the *Mean Absolute Error*, and finally the test data is prepared and fed to the model and results are printed to the `submission.csv` file. As expected, this simple model has a score of 2,314 on the training set (the score on the test set can only be computed by the competition's promoters), which denotes a really bad performance.

```
score = mean_absolute_error(y_train.values.flatten(), y_pred)
print(f'Score: {score:0.3f}')
```

```
submission = pd.read_csv('../input/sample_submission.csv', index_col='seg_id')
X_test = pd.DataFrame(columns=X_train.columns, dtype=np.float64, index=submission.index)
```

```
for seg_id in X_test.index:
    seg = pd.read_csv('../input/test/' + seg_id + '.csv')

    x = seg['acoustic_data'].values

    X_test.loc[seg_id, 'ave'] = x.mean()
    X_test.loc[seg_id, 'std'] = x.std()
    X_test.loc[seg_id, 'max'] = x.max()
    X_test.loc[seg_id, 'min'] = x.min()
```

```
X_test_scaled = scaler.transform(X_test)
submission['time_to_failure'] = svm.predict(X_test_scaled)
submission.to_csv('submission.csv')
```

Chapter 3

More advanced models comparison

Starting from the basic model presented in the previous section, for the purpose of this project activity we theorized and tested various approaches to solving the problem.

Lasso Elastic net Gaussian Probability Regression Isotonic Regression

3.1 First model: adding more features with Linear Regression

In order to make a step forward in the development of the regression model, our first attempt was to simply check out the performances obtained by a simple model with a few more features than the basic ones. In the attempt to study the impact of different features on the results, our choice for the model was the simple `Linear Regression` (implemented in `scikit-learn`), applied on a set of 12 features.

```
# Create a training file with simple derived features

rows = 150000
segments = int( np.floor(train.shape[0]) / rows)

X_train = pd.DataFrame(index=range(segments), dtype=np.float64,
                     columns=['ave', 'std', 'max', 'min', 'mad', 'kurt', 'skew', 'median', 'q01',
                            'q05', 'q95', 'q99'])

y_train = pd.DataFrame(index=range(segments), dtype=np.float64,
                     columns=['time_to_failure'])

for segment in tqdm(range(segments)):
    seg = train.iloc[segment*rows:segment*rows+rows]

```

```

    x = seg['acoustic_data']
    y = seg['time_to_failure'].values[-1]

    y_train.loc[segment, 'time_to_failure'] = y

    X_train.loc[segment, 'ave'] = x.mean()
    X_train.loc[segment, 'std'] = x.std()
    X_train.loc[segment, 'max'] = x.max()
    X_train.loc[segment, 'min'] = x.min()
    X_train.loc[segment, 'mad'] = x.mad()
    X_train.loc[segment, 'kurt'] = kurtosis(x)
    X_train.loc[segment, 'skew'] = skew(x)
    X_train.loc[segment, 'median'] = x.median()
    X_train.loc[segment, 'q01'] = np.quantile(x, 0.01)
    X_train.loc[segment, 'q05'] = np.quantile(x, 0.05)
    X_train.loc[segment, 'q95'] = np.quantile(x, 0.95)
    X_train.loc[segment, 'q99'] = np.quantile(x, 0.99)
```

In particular, we chose quite a standard set of features to add to the first four: `MAD` returns the Mean Absolute Deviation on the values (its accuracy is closely related to the Mean Squared Error, or `MSE`); *kurtosis* is a measure of the "tailedness" (or the shape) of the probability distribution of a real-valued random variable, calculated as the fourth standardized moment; *skewness* is a measure of the asymmetry of the probability distribution of a real-valued random variable, calculated as the third standardized moment; the *median* is the value separating the higher half from the lower half of the data; the *q-th quantiles* are cut points dividing the range of a probability distribution into intervals with the same probability: x is a q-th quantile for a variable X if $Pr[X < x] \leq q$.

The computed score of the so constructed model improves, even if slightly, the results of the naïve model, with a value of 2,251.

At this point, out of curiosity we took a look at the predicted `time_to_failure` resulting from the test data, and we noticed that there was a considerable number of negative values, evidently wrong (it should be remembered that they represent the time between the current segment and the next laboratory earthquake, which cannot be negative quantities).

Based on that observation, and given that the nature of the data is approximately symmetrical (see figure 1.4), we though about introducing a whole new set of features generated by the same computational functions applied on the absolute values of the dataset.

```
X_train = pd.DataFrame(index=range(segments), dtype=np.float64,
                     columns=['ave', 'std', 'max', 'min', 'mad', 'kurt', 'skew', 'median', 'q01',
                           'q05', 'q95', 'q99', 'abs_mean', 'abs_std', 'abs_max', 'abs_min', '
                          abs_mad', 'abs_kurt', 'abs_skew', 'abs_median', 'abs_q01', 'abs_q05',
                          'abs_q95', 'abs_q99'])
```

```
[...]
X_train.loc[segment, 'abs_mean'] = x.abs().mean()
X_train.loc[segment, 'abs_std'] = x.abs().std()
X_train.loc[segment, 'abs_max'] = x.abs().max()
X_train.loc[segment, 'abs_min'] = x.abs().min()
X_train.loc[segment, 'abs_mad'] = x.abs().mad()
X_train.loc[segment, 'abs_kurt'] = kurtosis(x.abs())
X_train.loc[segment, 'abs_skew'] = skew(x.abs())
X_train.loc[segment, 'abs_median'] = x.abs().median()
X_train.loc[segment, 'abs_q01'] = np.quantile(x.abs(), 0.01)
X_train.loc[segment, 'abs_q05'] = np.quantile(x.abs(), 0.05)
X_train.loc[segment, 'abs_q95'] = np.quantile(x.abs(), 0.95)
X_train.loc[segment, 'abs_q99'] = np.quantile(x.abs(), 0.99)
```

The results obtained in terms of score and predictions using this set of 24 values entailed another slight improvement, giving a value of 2,097 for the mean absolute error (our score), and fewer negative predictions in `submission.csv`.

After submitting our results to Kaggle's platform, the score of our kernel calculated by the system was 1,660.

3.2 Second model

3.3 Third model

Bibliography

[1] LANL Earthquake Prediction. 2019. URL: https://www.kaggle.com/c/LANL-Earthquake-Prediction.

[2] Allunia. Shaking Earth. 2019. URL: https://www.kaggle.com/allunia/shaking-earth.

[3] J. R. Leeman et al. Laboratory observations of slow earthquakes and the spectrum of tectonic fault slip modes. nat. commun. 7:11104 doi: 10.1038/ncomms11104. Technical report. 2016. URL: https://www.nature.com/articles/ncomms11104.

[4] inversion. Basic Feature Benchmark. 2019. URL: https://www.kaggle.com/inversion/basic-feature-benchmark.

[5] Gabriel Preda. LANL Earthquake EDA and Prediction. 2019. URL: https://www.kaggle.com/gpreda/lanl-earthquake-eda-and-prediction.

[6] Bertrand Rouet-Leduc, Claudia Hulbert, Nicholas Lubbers, Kipton Barros, Colin J. Humphreys, and Paul A. Johnson. Machine Learning Predicts Laboratory Earthquakes. Technical report. 2017. URL: https://doi.org/10.1002/2017GL074677.