Earthquake prediction model by using python

Phase 3: Topic:Loading and Preprocessing the Earthquake Prediction Model By using Python

Introduction:

* Machine learning has the ability to advance our knowledge of earthquakes and enable more accurate forecasting and catastrophe respon.
* It's crucial to remember that developing accurate and dependable prediction models for earthquakes still needs more study as it is a complicated and difficult topic.
* In order to anticipate earthquakes, machine learning may be used to examine seismic data trends. Seismometers capture seismic data, which may be used to spot changes to the earth's surface, like seismic waves brought on by earthquakes.
* Machine learning algorithms may utilize these patterns to forecast the risk of an earthquake happening in a certain region by studying these patterns and learning to recognize key traits that are linked to seismic activity.
* So we will be predicting the earthquake and Time, Latitude, and Longitude from previous data is not a trend that follows like other things. It is naturally occurring
* earthquake

The data are from an experiment conducted on rock in a double direct shear geometry subjected to bi-axial loading, a classic laboratory earthquake model.

Two fault gouge layers are sheared simultaneously while subjected to a constant normal load and a prescribed shear velocity. The laboratory faults fail in repetitive cycles of stick and slip that is meant to mimic the cycle of loading and failure on tectonic faults. While the experiment is considerably simpler than a fault in Earth, it shares many physical characteristics.

Los Alamos' initial work showed that the prediction of laboratory earthquakes from continuous seismic data is possible in the case of quasi-periodic laboratory seismic cycles.

Competition

In this competition, the team has provided a much more challenging dataset with considerably more aperiodic earthquake failures.  
Objective of the competition is to predict the failures for each test set.

**Prepare the data analysis**

## Load packages

Here we define the packages for data manipulation, feature engineering and model training.

unfold\_lessHide code

In [1]:

linkcode

import gc

import os

import time

import logging

import datetime

import warnings

import numpy as np

import pandas as pd

import seaborn as sns

import xgboost as xgb

import lightgbm as lgb

from scipy import stats

from scipy.signal import hann

from tqdm import tqdm\_notebook

import matplotlib.pyplot as plt

from scipy.signal import hilbert

from scipy.signal import convolve

from sklearn.svm import NuSVR, SVR

from catboost import CatBoostRegressor

from sklearn.kernel\_ridge import KernelRidge

from sklearn.metrics import mean\_squared\_error

from sklearn.preprocessing import LabelEncoder

from sklearn.metrics import mean\_absolute\_error

from sklearn.preprocessing import StandardScaler

from sklearn.linear\_model import LinearRegression

from sklearn.model\_selection import KFold,StratifiedKFold, RepeatedKFold

warnings.filterwarnings("ignore")

Load the data

Let's see first what files we have in input directory.

unfold\_lessHide code

In [2]:

IS\_LOCAL = False

if(IS\_LOCAL):

PATH="../input/LANL/"

else:

PATH="../input/"

os.listdir(PATH)

Out[2]:

['test', 'sample\_submission.csv', 'train.csv']

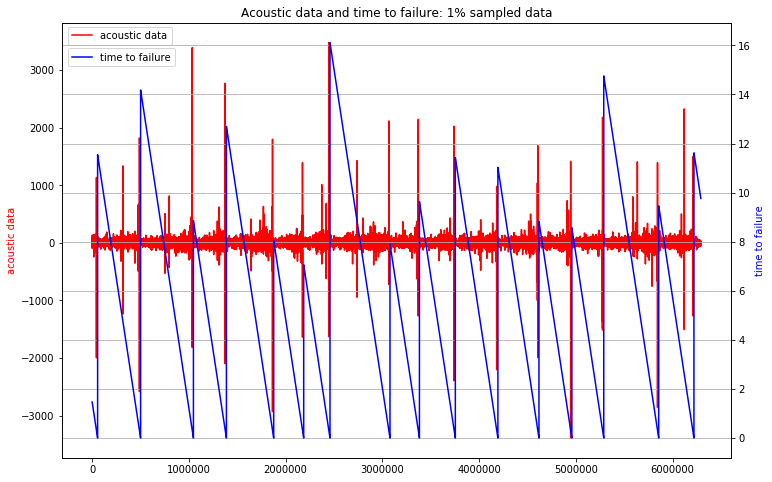
print("There are **{}** files in test folder".format(len(os.listdir(os.path.join(PATH, 'test' )))))

CPU times: user 2min 20s, sys: 14.1 s, total: 2min 34s

Wall time: 2min 35s

pd.options.display.precision = 15

train\_df.head(10)



train\_ad\_sample\_df = train\_df['acoustic\_data'].values[:6291455]

train\_ttf\_sample\_df = train\_df['time\_to\_failure'].values[:6291455]

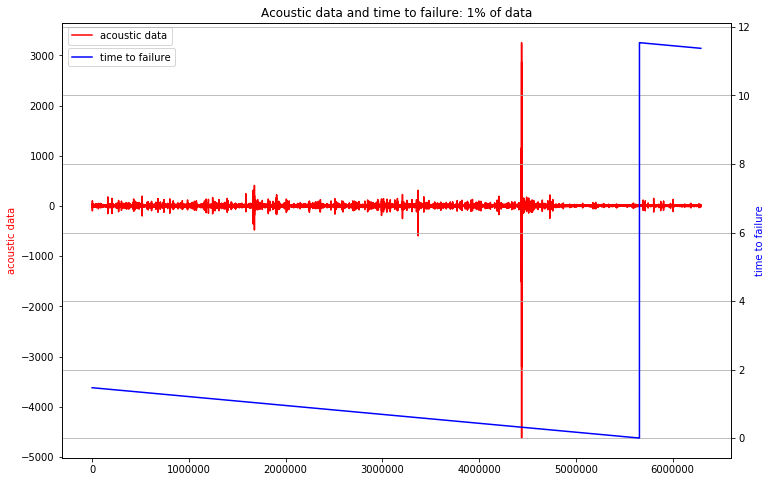
plot\_acc\_ttf\_data(train\_ad\_sample\_df, train\_ttf\_sample\_df, title="Acoustic data and time to failure: 1**% o**f data")

del train\_ad\_sample\_df

del train\_ttf\_sample\_df

# **Features engineering**

The test segments are 150,000 each.  
We split the train data in segments of the same dimmension with the test sets.



train\_X = pd.DataFrame(index=range(segments), dtype=np.float64)

train\_y = pd.DataFrame(index=range(segments), dtype=np.float64, columns=['time\_to\_failure'])

total\_mean = train\_df['acoustic\_data'].mean()

total\_std = train\_df['acoustic\_data'].std()

total\_max = train\_df['acoustic\_data'].max()

total\_min = train\_df['acoustic\_data'].min()

total\_sum = train\_df['acoustic\_data'].sum()

total\_abs\_sum = np.abs(train\_df['acoustic\_data']).sum()

unfold\_moreShow hidden code

train\_X = pd.DataFrame(index=range(segments), dtype=np.float64)

train\_y = pd.DataFrame(index=range(segments), dtype=np.float64, columns=['time\_to\_failure'])

total\_mean = train\_df['acoustic\_data'].mean()

total\_std = train\_df['acoustic\_data'].std()

total\_max = train\_df['acoustic\_data'].max()

total\_min = train\_df['acoustic\_data'].min()

total\_sum = train\_df['acoustic\_data'].sum()

total\_abs\_sum = np.abs(train\_df['acoustic\_data']).sum()

oof = np.zeros(len(scaled\_train\_X))

predictions = np.zeros(len(scaled\_test\_X))

feature\_importance\_df = pd.DataFrame()

*#run model*

for fold\_, (trn\_idx, val\_idx) **in** enumerate(folds.split(scaled\_train\_X,train\_y.values)):

strLog = "fold **{}**".format(fold\_)

print(strLog)

X\_tr, X\_val = scaled\_train\_X.iloc[trn\_idx], scaled\_train\_X.iloc[val\_idx]

y\_tr, y\_val = train\_y.iloc[trn\_idx], train\_y.iloc[val\_idx]

model = lgb.LGBMRegressor(\*\*params, n\_estimators = 20000, n\_jobs = -1)

model.fit(X\_tr,

y\_tr,

eval\_set=[(X\_tr, y\_tr), (X\_val, y\_val)],

eval\_metric='mae',

verbose=1000,

early\_stopping\_rounds=500)

oof[val\_idx] = model.predict(X\_val, num\_iteration=model.best\_iteration\_)

*#feature importance*

fold\_importance\_df = pd.DataFrame()

fold\_importance\_df["Feature"] = train\_columns

fold\_importance\_df["importance"] = model.feature\_importances\_[:len(train\_columns)]

fold\_importance\_df["fold"] = fold\_ + 1

feature\_importance\_df = pd.concat([feature\_importance\_df, fold\_importance\_df], axis=0)

*#predictions*

predictions += model.predict(scaled\_test\_X, num\_iteration=model.best\_iteration\_) / folds.n\_splits

fold 0

Training until validation scores don't improve for 500 rounds.

[1000] training's l1: 1.95653 valid\_1's l1: 2.25767

[2000] training's l1: 1.56603 valid\_1's l1: 2.13002

[3000] training's l1: 1.33595 valid\_1's l1: 2.10806

[4000] training's l1: 1.16028 valid\_1's l1: 2.10499

Early stopping, best iteration is:

[3692] training's l1: 1.21055 valid\_1's l1: 2.1045

fold 1

Training until validation scores don't improve for 500 rounds.

[1000] training's l1: 1.94954 valid\_1's l1: 2.27403

[2000] training's l1: 1.55888 valid\_1's l1: 2.14423

[3000] training's l1: 1.33327 valid\_1's l1: 2.12142

[4000] training's l1: 1.15832 valid\_1's l1: 2.11415

Early stopping, best iteration is:

[4353] training's l1: 1.10461 valid\_1's l1: 2.11341

fold 2

Training until validation scores don't improve for 500 rounds.

[1000] training's l1: 1.95751 valid\_1's l1: 2.28141

[2000] training's l1: 1.57091 valid\_1's l1: 2.11881

[3000] training's l1: 1.34303 valid\_1's l1: 2.08681

[4000] training's l1: 1.16558 valid\_1's l1: 2.08075

[5000] training's l1: 1.01986 valid\_1's l1: 2.08095

Early stopping, best iteration is:

[4543] training's l1: 1.08318 valid\_1's l1: 2.08006

fold 3

Training until validation scores don't improve for 500 rounds.

[1000] training's l1: 1.97583 valid\_1's l1: 2.13799

[2000] training's l1: 1.57798 valid\_1's l1: 2.03289

[3000] training's l1: 1.34362 valid\_1's l1: 2.02755

Early stopping, best iteration is:

[3226] training's l1: 1.29993 valid\_1's l1: 2.02722

fold 4

Training until validation scores don't improve for 500 rounds.

[1000] training's l1: 1.94634 valid\_1's l1: 2.26153

[2000] training's l1: 1.55907 valid\_1's l1: 2.11486

[3000] training's l1: 1.33059 valid\_1's l1: 2.09017

[4000] training's l1: 1.15548 valid\_1's l1: 2.08648

Early stopping, best iteration is:

[3918] training's l1: 1.16854 valid\_1's l1: 2.08631

Necessary step to follow:

1.Import Libraries:

Start by importing the necessary libraries:

Program:

import pandas as pd

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

2.Load the Dataset:

Load your dataset into a Pandas DataFrame. You can typically find

house price datasets in CSV format, but you can adapt this code to other

formats as needed.

Program:

df = pd.read\_csv(' E:\USA\_Housing.csv ')

Pd.read()

3. Exploratory Data Analysis (EDA):

Perform EDA to understand your data better. This includes

checking for missing values, exploring the data's statistics, and

visualizing it to identify patterns.

Program:

# Check for missing values

print(df.isnull().sum())

# Explore statistics

print(df.describe())

# Visualize the data (e.g., histograms, scatter plots, etc.)

Feature Engineering:

Depending on your dataset, you may need to create new features or

transform existing ones. This can involve one-hot encoding categorical

variables, handling date/time data, or scaling numerical features.

Program:

# Example: One-hot encoding for categorical variables

df = pd.get\_dummies(df, columns=[' Avg. Area Income ', ' Avg. Area

House Age '])

5. Split the Data:

Split your dataset into training and testing sets. This helps you evaluate

your model's performance later.

X = df.drop('price', axis=1) # Features

y = df['price'] # Target variable

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2,

random\_state=42)

6. Feature Scaling:

Apply feature scaling to normalize your data, ensuring that all

features have similar scales. Standardization (scaling to mean=0 and

std=1) is a common choice.

Program:

scaler = StandardScaler()

X\_train = scaler.fit\_transform(X\_train)

X\_test = scaler.transform(X\_test)

Importance of loading and processing dataset:

Loading and preprocessing the dataset is an important first step in

building any machine learning model. However, it is especially

important for house price prediction models, as house price datasets are

often complex and noisy.

By loading and preprocessing the dataset, we can ensure that the

machine learning algorithm is able to learn from the data effectively and

accurately.

Challenges involved in loading and preprocessing a house price

dataset;

There are a number of challenges involved in loading and preprocessing

a house price dataset, including:

 Handling missing values:

House price datasets often contain missing values, which can

be due to a variety of factors, such as human error or incomplete data

collection. Common methods for handling missing values include

dropping the rows with missing values, imputing the missing values with

the mean or median of the feature, or using a more sophisticated method

such as multiple imputation.

 Encoding categorical variables:

House price datasets often contain categorical features, such as the

type of house, the neighborhood, and the school district. These features

need to be encoded before they can be used by machine learning models.

One common way to encode categorical variables is to use one-hot

encoding.

 Scaling the features:

It is often helpful to scale the features before training a

machine learning model. This can help to improve the performance of

the model and make it more robust to outliers. There are a variety of

ways to scale the features, such as min-max scaling and standard scaling.

 Splitting the dataset into training and testing sets:

Once the data has been pre-processed, we need to split the

dataset into training and testing sets. The training set will be used to

train the model, and the testing set will be used to evaluate the

performance of the model on unseen data. It is important to split the

dataset in a way that is representative of the real world distribution of the

data.

How to overcome the challenges of loading and preprocessing a

house price dataset:

There are a number of things that can be done to overcome the

challenges of loading and preprocessing a house price dataset, including:

 Use a data preprocessing library:

There are a number of libraries available that can help with data

preprocessing tasks, such as handling missing values, encoding

categorical variables, and scaling the features.

 Carefully consider the specific needs of your model:

The best way to preprocess the data will depend on the specific

machine learning algorithm that you are using. It is important to

carefully consider the requirements of the algorithm and to preprocess

the data in a way that is compatible with the algorithm.

Validate the preprocessed data:

It is important to validate the preprocessed data to ensure that it is

in a format that can be used by the machine learning algorithm and that

it is of high quality. This can be done by inspecting the data visually or

by using statistical methods.

1.Loading the dataset:

 Loading the dataset using machine learning is the process of bringing

the data into the machine learning environment so that it can be used

to train and evaluate a model.

 The specific steps involved in loading the dataset will vary depending

on the machine learning library or framework that is being used.

However, there are some general steps that are common to most

machine learning frameworks:

2.Identify the dataset:

The first step is to identify the dataset that you want to load. This

dataset may be stored in a local file, in a database, or in a cloud storage

service.

b.Load the dataset:

Once you have identified the dataset, you need to load it into the

machine learning environment. This may involve using a built-in

function in the machine learning library, or it may involve writing your

own code.

c.Preprocess the dataset:

Once the dataset is loaded into the machine learning environment,

you may need to preprocess it before you can start training and

evaluating your model. This may involve cleaning the data, transforming

the data into a suitable format, and splitting the data into training and

test sets.

Preprocess the

dataset

Load the dataset

Identify the

dataset

Loading the

dataset

Here, how to load a dataset using machine learning in Python

**CONCLUSION:**

When comparing two models, both the mean squared error (MSE) and R-squared (R2) score can be used to evaluate the performance of the models.

In general, a model with a lower MSE and a higher R2 score is considered a better model. This is because the MSE measures the average difference between the predicted and actual values, and a lower MSE indicates that the model is making more accurate predictions. The R2 score measures the proportion of the variance in the target variable that is explained by the model, and a higher R2 score indicates that the model is able to explain more of the variability in the target variable.

From the results of this project we can conclude that random forest is the most accurate model for predicting the magnitude of Earthquake compared to all other models used in this project.

However, it's important to keep in mind that the relative importance of MSE and R2 score may vary depending on the specific problem and the context in which the models are being used. For example, in some cases, minimizing the MSE may be more important than maximizing the R2 score, or vice versa. It's also possible that one model may perform better on one metric and worse on another, so it's important to consider both metrics together when evaluating the performance of the models.