AI\_phase 3

Earthquake prediction model using Python

Start building the earthquake prediction model by loading and pre-processing the dataset.

To start building an earthquake prediction model, you should follow these steps for loading and preprocessing the dataset:

Dataset Acquisition: First, acquire the earthquake dataset. You can find such datasets from sources like USGS (United States Geological Survey) or other reputable sources.

Data Loading: Use a programming language like Python and libraries like Pandas to load the dataset into your project.

For example:

import pandas as pd

# Load the dataset

data = pd.read\_csv(“earthquake\_data.csv”)

Data Exploration: Begin by exploring the dataset to understand its structure and the types of data it contains. You can use methods like head(), info(), and describe() to get a preliminary overview.

Data Preprocessing: Perform necessary data preprocessing tasks such as handling missing values, removing duplicates, and handling outliers. You might need to convert data types or normalize features as well.

Feature Selection/Engineering: Identify the relevant features for earthquake prediction. You might also need to engineer new features from the existing data if it helps improve the model’s accuracy.

Data Splitting: Split the dataset into training and testing sets to evaluate your model’s performance.

Data Visualization: Create visualizations to gain insights into the data and detect patterns that could be useful for prediction. Matplotlib or Seaborn are handy libraries for data visualization in Python.

Building a basic earthquake prediction model using Python and scikit-learn.

import numpy as np

from sklearn.model\_selection import cross\_val\_score

from catboost import CatBoostRegressor

# set output float precision

np.set\_printoptions(precision=3)

# init model

model = CatBoostRegressor(random\_seed=0, verbose=False)

# calculate mae on folds

mae = cross\_val\_score(model, data[best\_features], data['target'],

cv=5, scoring='neg\_mean\_absolute\_error', n\_jobs=8)

# print the results

print('folds: {}'.format(abs(mae)))

print('total: {:.3f}'.format(np.mean(abs(mae))))

import random

class Chromosome(object):

def \_\_init\_\_(self, genes, size):

self.genes = random.sample(genes, size)

from deap import base, creator, tools

def init\_individual(ind\_class, genes=None, size=None):

return ind\_class(genes, size)

genes = [

column for column in train.columns

if column not in ['target', 'seg\_id']

]

# setting individual creator

creator.create('FitnessMin', base.Fitness, weights=(-1,))

creator.create('Individual', Chromosome, fitness=creator.FitnessMin)

# register callbacks

toolbox = base.Toolbox()

toolbox.register(

'individual', init\_individual, creator.Individual,

genes=genes, size=n\_features)

toolbox.register(

'population', tools.initRepeat, list, toolbox.individual)

# raise population

pop = toolbox.population(50)

toolbox.register('mate', tools.cxTwoPoint)

def mutate(individual, genes=None, pb=0):

# maximal amount of mutated genes

n\_mutated\_max = max(1, int(len(individual) \* pb))

# generate the random amount of mutated genes

n\_mutated = random.randint(1, n\_mutated\_max)

# select random genes which need to be mutated

mutated\_indexes = random.sample(

[index for index in range(len(individual.genes))], n\_mutated)

# mutation

for index in mutated\_indexes:

individual[index] = random.choice(genes)

return individual,

toolbox.register('mutate', mutate, genes=genes, pb=0.2)

from catboost import CatBoostRegressor

from sklearn.model\_selection import cross\_val\_score

model = CatBoostRegressor(

iterations=60, learning\_rate=0.2, random\_seed=0, verbose=False)

def evaluate(individual, model=None, train=None, n\_splits=5):

mae\_folds = cross\_val\_score(

model,

train[individual.genes],

train['target'],

cv=n\_splits,

scoring='neg\_mean\_absolute\_error')

return abs(mae\_folds.mean()),

toolbox.register(

'evaluate', evaluate, model=model, train=train, n\_splits=5)

from operator import attrgetter

def select\_best(individuals, k, fit\_attr='fitness'):

return sorted(set(individuals), key=attrgetter(fit\_attr), reverse=True)[:k]

toolbox.register('select', select\_best)

hof = tools.HallOfFame(5)

from deap import algorithms

# mu: the number of individuals to select for the next generation

# lambda: the number of children to produce at each generation

# cxpb: the probability that offspring is produced by crossover

# mutpb: the probability that offspring is produced by mutation

# ngen: the number of generations

algorithms.eaMuPlusLambda(

pop, toolbox,

mu=10, lambda\_=30, cxpb=0.2, mutpb=0.8,

ngen=50, stats=stats, halloffame=hof, verbose=True)