Code

Issues

Pull requests

More

Kaggle LANL Earthquake Prediction challenge, Genetic Algorithm (DEAP) + CatboostRegressor, private score 2.425 (31 place)

viktorsapozhok.github.io/deap-genetic-algorithm/

License

MIT license

17 stars

4 forks

2 watching

Activity

Public repository

viktorsapozhok/earthquake-prediction

Branches

Tags

Latest commit

@viktorsapozhok

viktorsapozhok

…

on Jan 25, 2021

Git stats

Files

README.md

LANL Earthquake Prediction

This repository presents an approach used for solving Kaggle LANL Earthquake Prediction Challenge.

For feature engineering we used this kernel, slightly modified for adding some spectral features. The initial training set /data/train.csv contains 4194 rows (one row for each segment) and 1496 columns (features). We applied genetic algorithm with CatboostRegressor for fitness evaluation to implement a feature selection. Based on the GA's results, we selected 15 features and trained the model using CatboostRegressor with default parameters.

Project structure

.

├── ...

├── data

├── train.csv # Original training set decomposed into feature set

├── test.csv # Testing signal decomposed into feature set

└── results.csv # Modeling results prepared for submission

│── notebooks

└── earthquake.ipynb # Misc

├── earthquake

├── config.py # Configuration parameters

├── ga.py # GA for feature selection

├── generator.py # Feature engineering

├── submission.py # Make prediction and prepare file for submission

└── utils.py # Helpers

└── ...

Note, that train and test sets in data directory contain only 200 of 1496 features to reduce the file size.

How to run

First, clone the repository and install it from setup file:

$ git clone https://github.com/viktorsapozhok/earthquake-prediction.git

$ cd earthquake-prediction

$ pip install --editable .

To start the genetic algorithm implementing feature selection, launch ga.py script from project's root directory:

$ python earthquake/ga.py

Feature engineering

The initial acoustic signal is decomposed into segments with 150000 rows per segment, which suggests that the training dataset has 4194 rows. Features are calculated as aggregations over segments. For more details see, for example, here and here.

Baseline model

Before we start with the feature selection, we calculate feature importance as it is explained here and train the baseline model on the 15 most important features.

from earthquake import config, utils

# load training set

data = utils.read\_csv(config.path\_to\_train)

# create list of features

features = [column for column in data.columns if column not in ['target', 'seg\_id']]

# display importance

best\_features = utils.feature\_importance(data[features], data['target'], n\_best=15, n\_jobs=8)

List of 15 most important features.

Imp | Feature

0.11 | mfcc\_5\_avg

0.09 | mfcc\_15\_avg

0.07 | percentile\_roll\_std\_5\_window\_50

0.06 | percentile\_roll\_std\_10\_window\_100

0.06 | mfcc\_4\_avg

0.03 | percentile\_roll\_std\_20\_window\_500

0.03 | percentile\_roll\_std\_25\_window\_500

0.02 | percentile\_roll\_std\_25\_window\_100

0.02 | percentile\_roll\_std\_20\_window\_1000

0.02 | percentile\_roll\_std\_20\_window\_10

0.02 | percentile\_roll\_std\_25\_window\_1000

0.01 | percentile\_roll\_std\_10\_window\_500

0.01 | percentile\_roll\_std\_10\_window\_50

0.01 | percentile\_roll\_std\_50\_window\_50

0.01 | percentile\_roll\_std\_40\_window\_1000

We train the model using CatboostRegressor with default parameters and evaluate the performance with a stratified KFold (5 folds) cross-validation.

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))))

CatboostRegressor (without any tuning) trained on 15 features having highest importance score demonstrates mean average error 2.064.

folds: [1.982 2.333 2.379 1.266 2.362]

total: 2.064

Feature selection

To avoid a potential overfitting, we employ a genetic algorithm for feature selection. The genetic context is pretty straightforward. We suppose that the list of features (without duplicates) is the chromosome, whereas each gene represents one feature. n\_features is the input parameter controlling the amount of genes in the chromosome.

import random

class Chromosome(object):

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

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

We generate the population with 50 chromosomes, where each gene is generated as a random choice from initial list of features (1496 features). To accelerate the performance, we also add to population the feature set used in the baseline model.

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)

Standard two-point crossover operator is used for crossing two chromosomes.

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

To implement a mutation, we first generate a random amount of genes (> 1), which needs to be mutated, and then mutate these genes in order that the chromosome doesn't contain two equal genes.

Note, that mutation operator must return a tuple.

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)

For fitness evaluation we use lightened version of CatboostRegressor with decreased number of iterations and increased learning rate. Note, that fitness evaluator must also return a tuple.

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)

We register elitism operator to select best individuals to the next generation. The amount of the best individuals is controlling by the parameter mu in the algorithm. To prevent populations with many duplicate individuals, we overwrite the standard selBest operator.

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)

To keep track of the best individuals, we introduce a hall of fame container.

hof = tools.HallOfFame(5)

Finally, we put everything together and launch eaMuPlusLambda evolutionary algorithm. Here we set cxpb=0.2, the probability that offspring is produced by the crossover, and mutpb=0.8, the probability that offspring is produced by mutation. Mutation probability is intentionally increased to prevent a high occurrence of identical chromosomes produced by the crossover.

As a result, we get the list of 15 best features selected into the model.

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)

Here is the list of 15 features accumulated in the best chromosome after 50 generations.

1. ffti\_av\_change\_rate\_roll\_mean\_1000

2. percentile\_roll\_std\_30\_window\_50

3. skew

4. percentile\_roll\_std\_10\_window\_100

5. percentile\_roll\_std\_30\_window\_50

6. percentile\_roll\_std\_20\_window\_1000

7. ffti\_exp\_Moving\_average\_30000\_mean

8. range\_3000\_4000

9. max\_last\_10000

10. mfcc\_4\_avg

11. fftr\_percentile\_roll\_std\_80\_window\_10000

12. percentile\_roll\_std\_1\_window\_100

13. ffti\_abs\_trend

14. av\_change\_abs\_roll\_mean\_50

15. mfcc\_15\_avg