# 应用开发指导说明书

四川大学

2022/10/20

## 1 应用案例

本项目得到的2500条NAS-Bench-101数据和2500条NAS-Bench-201数据可以用于构建性能预测器。因此，在本应用案例当中，我们用一个简单的性能预测器，即随机森林模型，作为例子，展示如何应用我们的2500个NAS-Bench-101架构构建性能预测器，并用该性能预测器评估架构。

### 1.1 设计过程

应用案例的设计过程如图1-1所示，我们将评估得到的2500条架构数据中的2000条架构数据编码得到性能预测器的训练集。除此之外，我们将另外500条架构数据编码后形成性能预测器的测试集。我们用训练集训练随机森林预测器，并将随机森林预测器在测试集上进行测试，得到性能预测器的测试准确度。

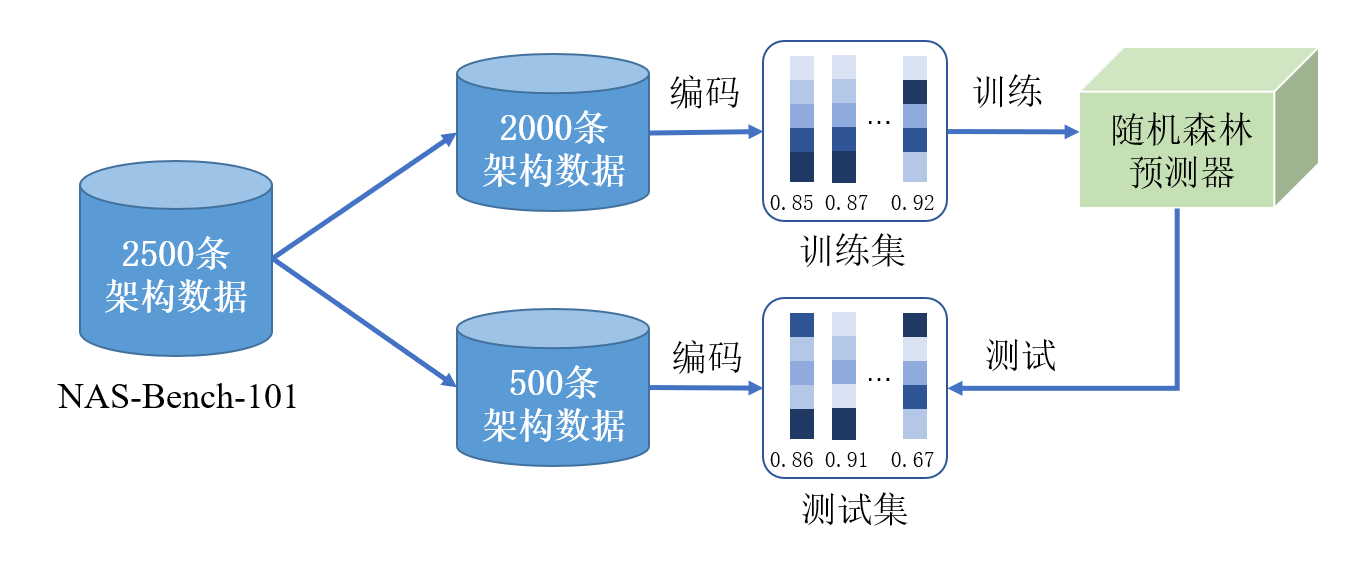


图1-1. 应用案例设计过程

我们的编码方法将架构数据中的邻接矩阵M化为一维的向量；同时将节点操作L用one-hot编码后再化为一维的向量，然后连接形成最后的编码。性能预测器采用经典的随机森林预测器。除此之外，我们采用均方误差作为损失函数。

### 1.2 开发过程

我们将从数据读取，模型构建，模型训练三个方面来描述我们的开发过程。

**1.2.1 数据集读取及划分**

加载数据集的过程如下所示：

def dataset\_load(train\_num, test\_num):

    #加载数据集中的key及对应的评估信息。

    model\_id\_regex="^"

    regex = re.compile(model\_id\_regex)

    with tf.gfile.Open("data/data.json") as f:

        exist\_models = json.load(f)

    exist\_keys = [key for key in exist\_models.keys() if regex.match(key)]

    random.shuffle(exist\_keys)

    #分配训练集的key和测试集的key

    train\_keys = exist\_keys[0:train\_num]

    test\_keys = exist\_keys[train\_num:train\_num+test\_num]

    #获得key对应的架构的邻接矩阵和节点操作

    with tf.gfile.Open("data/generated\_graphs.json") as f:

        print("exist!")

        all\_models = json.load(f)

    #分配训练集的邻接矩阵和节点操作以及标签（即性能）

    train\_matrix = []

    train\_ops = []

    train\_label = []

    for i in range(len(train\_keys)):

        key = train\_keys[i]

        acc = exist\_models[key]['evaluation\_results'][2]['test\_accuracy']

        matrix = all\_models[key][0]

        ops = all\_models[key][1]

        train\_matrix.append(matrix)

        train\_ops.append(ops)

        train\_label.append(acc)

    #分配测试集的邻接矩阵和节点操作以及标签（即性能）

    test\_matrix = []

    test\_ops = []

    test\_label = []

    for i in range(len(test\_keys)):

        key = test\_keys[i]

        acc = exist\_models[key]['evaluation\_results'][2]['test\_accuracy']

        matrix = all\_models[key][0]

        ops = all\_models[key][1]

        test\_matrix.append(matrix)

        test\_ops.append(ops)

        test\_label.append(acc)

    return train\_matrix, train\_ops, train\_label, test\_matrix, test\_ops, test\_label

数据集的编码方法如下代码所示：

def encode(adjecent\_matrix, module\_operations):

    #填充架构为相同大小的邻接矩阵和相同长度的节点操作列表

    padding\_matrix, module\_operations = padding\_zero\_in\_matrix(adjecent\_matrix, module\_operations)

    #去除节点操作列表的in节点和out节点

    module\_operations.remove(-1)

    module\_operations.remove(-2)

    #将节点操作列表变成one-hot矩阵

    operations = []

    for i in range(0,len(module\_operations)):

        ops = [ 0 for i in range(0,5)]

        if i < 3:

            ops[i] = 1

            operations.append(ops)

        else:

            operations.append(ops)

    operations = np.array(operations)

    #将邻接矩阵和节点操作列表化为一维并连接

    adj = list(np.ndarray.flatten(padding\_matrix))

    operation = list(np.ndarray.flatten(operations))

    code = np.append(adj,operation)

    return code

编码训练集和测试集中的所有架构：

def encode\_data(train\_matrix, train\_ops, train\_label, test\_matrix, test\_ops, test\_label):

    x\_train = []

    for i in range(0,len(train\_matrix)):

        x\_train.append(encode(train\_matrix[i],train\_ops[i]))

    x\_test = []

    for i in range(0,len(test\_matrix)):

        x\_test.append(encode(test\_matrix[i],test\_ops[i]))

    return x\_train, train\_label, x\_test, test\_label

**1.2.2模型构建**

调用sklearn包，获得随机森林模型：

def get\_model():

model\_random\_forest\_regressor = ensemble.RandomForestRegressor(n\_estimators=230,max\_features='auto')

return model\_random\_forest\_regressor

该随机森林模型应用了230颗树，自动选取最大特征数。

**1.2.3模型训练**

训练模型并获得在测试集上的ktau值和R2分数来评估模型

def try\_model(x\_train, y\_train, x\_test, y\_test, model):

model.fit(x\_train, y\_train)

result = model.predict(x\_test)

result = list(result)

score = r2\_score(y\_test, result)

result\_arg = np.argsort(result)

y\_test\_arg = np.argsort(y\_test)

result\_rank = np.zeros(len(y\_test\_arg))

y\_test\_rank = np.zeros(len(y\_test\_arg))

for i in range(len(y\_test\_arg)):

result\_rank[result\_arg[i]] = i

y\_test\_rank[y\_test\_arg[i]] = i

KTau, \_ = kendalltau(result\_rank, y\_test\_rank)

print('KTau: {:}, R2score: {:}'.format(KTau, score))

## 2 二次开发指导

本项目主要为构建性能预测器提供数据，因此为了便于二次开发，我们主要提供数据获取的接口。并提供了构建预测器的代码示例。

### 2.1 数据获取接口

函数： dataset\_load(data\_dir, graph\_dir, train\_num, test\_num)

输入：

* data\_dir: 架构key及对应评估数据的存储地址。默认为：“data/data.json”
* graph\_dir：架构key及对应邻接矩阵和节点操作列表的存储地址。默认为：“data/generate\_graph.json”
* train\_num:训练集的数目
* test\_num:测试集数目

输出：

train\_matrix：训练集邻接矩阵

train\_ops：训练集节点操作列表。

train\_label：训练集标签。

test\_matrix：测试集邻接矩阵

test\_ops：测试集节点操作列表。

test\_label：测试集标签。

### 2.2 示例代码

from nasbench import api

import tensorflow as tf

import json

import re

import numpy as np

import random

from sklearn.metrics import r2\_score

from scipy.stats import kendalltau

from sklearn import ensemble

#邻接矩阵填充

def padding\_zero\_in\_matrix(adjecent\_matrix, module\_operations):

len\_operations = len(module\_operations)

if len\_operations != 7:

# if the operations is less than 7

for j in range(len\_operations, 7):

module\_operations.insert(j - 1, 3)

# print(important\_metrics[i]['fixed\_metrics']['module\_operations'])

padding\_matrix = np.insert(adjecent\_matrix, len\_operations - 1,

np.zeros([7 - len\_operations, len\_operations]), axis=0)

padding\_matrix = np.insert(padding\_matrix, [len\_operations - 1], np.zeros([7, 7 - len\_operations]), axis=1)

else:

padding\_matrix = np.array(adjecent\_matrix)

return padding\_matrix, module\_operations

#编码方法

def encode(adjecent\_matrix, module\_operations):

#填充架构为相同大小的邻接矩阵和相同长度的节点操作列表

padding\_matrix, module\_operations = padding\_zero\_in\_matrix(adjecent\_matrix, module\_operations)

#去除节点操作列表的in节点和out节点

module\_operations.remove(-1)

module\_operations.remove(-2)

#将节点操作列表变成one-hot矩阵

operations = []

for i in range(0,len(module\_operations)):

ops = [ 0 for i in range(0,5)]

if i < 3:

ops[i] = 1

operations.append(ops)

else:

operations.append(ops)

operations = np.array(operations)

#将邻接矩阵和节点操作列表化为一维并连接

adj = list(np.ndarray.flatten(padding\_matrix))

operation = list(np.ndarray.flatten(operations))

code = np.append(adj,operation)

return code

#数据加载

def dataset\_load(data\_dir="data/data.json", graph\_dir="data/generate\_graphs.json", train\_num=2000, test\_num=500):

#加载数据集中的key及对应的评估信息。

model\_id\_regex="^"

regex = re.compile(model\_id\_regex)

with tf.gfile.Open(data\_dir) as f:

exist\_models = json.load(f)

exist\_keys = [key for key in exist\_models.keys() if regex.match(key)]

random.shuffle(exist\_keys)

#分配训练集的key和测试集的key

train\_keys = exist\_keys[0:train\_num]

test\_keys = exist\_keys[train\_num:train\_num+test\_num]

#获得key对应的架构的邻接矩阵和节点操作

with tf.gfile.Open(graph\_dir) as f:

print("exist!")

all\_models = json.load(f)

#分配训练集的邻接矩阵和节点操作以及标签（即性能）

train\_matrix = []

train\_ops = []

train\_label = []

for i in range(len(train\_keys)):

key = train\_keys[i]

acc = exist\_models[key]['evaluation\_results'][2]['test\_accuracy']

matrix = all\_models[key][0]

ops = all\_models[key][1]

train\_matrix.append(matrix)

train\_ops.append(ops)

train\_label.append(acc)

#分配测试集的邻接矩阵和节点操作以及标签（即性能）

test\_matrix = []

test\_ops = []

test\_label = []

for i in range(len(test\_keys)):

key = test\_keys[i]

acc = exist\_models[key]['evaluation\_results'][2]['test\_accuracy']

matrix = all\_models[key][0]

ops = all\_models[key][1]

test\_matrix.append(matrix)

test\_ops.append(ops)

test\_label.append(acc)

return train\_matrix, train\_ops, train\_label, test\_matrix, test\_ops, test\_label

#获取模型

def get\_model():

model\_random\_forest\_regressor = ensemble.RandomForestRegressor(n\_estimators=230,max\_features='auto')

return model\_random\_forest\_regressor

#数据编码

def encode\_data(train\_matrix, train\_ops, train\_label, test\_matrix, test\_ops, test\_label):

x\_train = []

for i in range(0,len(train\_matrix)):

x\_train.append(encode(train\_matrix[i],train\_ops[i]))

x\_test = []

for i in range(0,len(test\_matrix)):

x\_test.append(encode(test\_matrix[i],test\_ops[i]))

return x\_train, train\_label, x\_test, test\_label

#训练模型

def try\_model(x\_train, y\_train, x\_test, y\_test, model):

model.fit(x\_train, y\_train)

result = model.predict(x\_test)

result = list(result)

score = r2\_score(y\_test, result)

result\_arg = np.argsort(result)

y\_test\_arg = np.argsort(y\_test)

result\_rank = np.zeros(len(y\_test\_arg))

y\_test\_rank = np.zeros(len(y\_test\_arg))

for i in range(len(y\_test\_arg)):

result\_rank[result\_arg[i]] = i

y\_test\_rank[y\_test\_arg[i]] = i

KTau, \_ = kendalltau(result\_rank, y\_test\_rank)

print('KTau: {:}, R2score: {:}'.format(KTau, score))

if \_\_name\_\_ == '\_\_main\_\_':

data\_dir = "data/data.json"

graph\_dir = "data/generated\_graphs.json"

train\_matrix, train\_ops, train\_label, test\_matrix, test\_ops, test\_label = dataset\_load(data\_dir, graph\_dir, 2000, 500)

x\_train, y\_train, x\_test, y\_test = encode\_data(train\_matrix, train\_ops, train\_label, test\_matrix, test\_ops, test\_label)

model = get\_model()

try\_model(x\_train, y\_train, x\_test, y\_test, model)